Design Time Detection Of Architectural Mismatches in Service Oriented Architectures

Thesis by

Carl J. Gamble

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

School of Computing Science,
Newcastle University,
Newcastle upon Tyne, UK

July 2011

For my girls, Evie and Lily.

Evie's laughter reminded me what was important and gave me inspiration when things were tough; and while Lily put in an appearance only a few days ago, her imminent arrival was a great reason to finish.

X

Acknowledgements

The sign-off of the final corrections to this thesis marks the end of a very long journey that has mostly been very enjoyable and certainly much better than having a real job! ;) And while the contributions and technical content included in the many many pages that follow [1] are certainly mine, I am quite positive that they would not be here or would be far more poor without the support of a few special people. So it is with great pleasure that the final words I will write here will recognise at least some of them.

I will start with my long suffering Ph.D. supervisor, Cristina Gacek. Somehow, through a combination of listening, guiding, commenting, encouragement and the occasional good hard push, she managed to get me through the Ph.D. I have thoroughly enjoyed the experience of working with her and sincerely hope that I have opportunity to do so again.

Thanks must also go to my thesis committee, Alexander Romanovsky, John Fitzgerald and Aad van Morsel for their sound advice and for helping to keep the scope of the work down to manageable size when I was getting a little ambitious towards the end.

My examiners, Tom Anderson and Michel Wermelinger are also deserving of praise. I actually rather enjoyed the discussions that took place during the viva and I think that the thesis is far better after implementing the changes they laid out for me.

I also need to thank Tom for helping to dig me out of deep holes when Brian Randell asked hard questions in presentations, but at the same time to extend thanks to Brian for asking the questions in the first place, these always caused me to think of the work in different ways.

I would also like to thank Bradley Schmerl of Carnegie Mellon University for correcting the bugs I found in ACME Studio and also for his assistance when I could not get my external analysis working. Related to this is my gratitude to Jeremy Bryans for patiently listening to my questions and ideas regarding the use of CSP in this work.

Thanks are also due to Steve Riddle for allowing me the time to complete the thesis while working for him as an RA. It would not have been possible to complete the work without his cooperation.

I also want to thank my fellow Ph.D. students. The gaming, lunch-time/coffee-break/any-other-time-we-felt-like-it chats and other good times all played a big part in keeping me sane!

[1] Sorry about the size of the thesis, I apparently don't know when to stop! ;)

Last, but by no means least, I come to my family. I want to thank my mother and father for their unshakable belief that I would get through the Ph.D., this is rather more confidence than I had at the time. Finally I want to thank my wife Emma and my little girl Evie. Without Emma's help and support I would not have been able to even consider doing the Ph.D. and without her insistence on holidays and family days out would have been in danger of losing touch with the world outside of the office. Evie's help was simply being her cheery, smiley, fun self and always being ready to play. Our trips to the park and to the beach were instrumental in helping me to relax and keep a perspective of what really matters at the end of the day.

I am sure I will have forgotten some people or some help, so if you are one of those people then please give yourself a pat on the back and say thanks to yourself from me :D.

Cheers!

Carl.

Publications

Aspects of this work have appeared in the following publications.

Journal

C. Gacek and C. Gamble. Mismatch avoidance in web services software architectures. *Journal of Universal Computer Science*, 14(8):1285 – 1313, 2008.

Conference

Carl Gamble, Detecting Architectural Mismatches Between Web Services. In *DSN '07: Supplemental Volume, 37th IEEE/IFIP International Conference on Dependable Systems and Networks,* pages 315–317, 25-28 June, 2007, IEEE Computer Society, 2007.

Workshop

Carl Gamble, Architectural Mismatch in Web Services. In *IRTG '06: Proceedings of the International Research Training Groups Workshop,* pages 52–53, 6-8 November, 2006, GITO-Verlag, Berlin, 2006.

Technical Reports

Carl Gamble and Cristina Gacek. Mismatch Avoidance in Web Services Based Software Architectures. Technical Report CS-TR-1079, Newcastle University, Newcastle upon Tyne, United Kingdom, 2008.

Carl Gamble and Cristina Gacek. Minimal Web Services Style Architectural Style Description and Example Instantiation. Technical Report CS-TR-1078, Newcastle University, Newcastle upon Tyne, United Kingdom, 2008.

Carl Gamble and Cristina Gacek. Mismatch Avoidance with Web Services. Technical Report CS-TR-1061, Newcastle University, Newcastle upon Tyne, United Kingdom, 2007.

Carl Gamble. Detecting Architectural Mismatches Between Web Services. Technical Report CS-TR-1019, Newcastle University, Newcastle upon Tyne, United Kingdom, 2007.

Carl Gamble. A minimal web service architectural style. Technical Report CS-TR-1015, Newcastle University, Newcastle upon Tyne, United Kingdom, 2007.

Abstract

Service Oriented Architecture (SOA) is a software component paradigm that has the potential to allow for flexible systems that are loosely coupled to each other. They are discoverable entities that may be bound to at run time by a client who is able to use the service correctly by referring to the service's description documents.

Assumptions often have to be made in any design process if the problem domain is not fully specified. If those decisions are about the software architecture of that component and it is inserted into a system with differing and incompatible assumptions then we say that an architectural mismatch exists.

Architectural styles are a form of software reuse. They can simply be used by referring to a name such as “client-server” or “pipe and filter”, where these names may conjure up topologies and expected properties in the architects mind. They can also however be more rigorously defined given the right software environment. This can lead to a vocabulary of elements in the system, defined properties of those elements along with rules and analysis to either show correctness of an implementation or reveal some emergent property of the whole.

SOA includes a requirement that the service components make available descriptions of themselves, indicating how they are to be used. With this in mind and assuming we have a suitable description of the client application it should be the case that we can detect architectural mismatches when designing a new system. Here designing can range from organising a set of existing components into a novel configuration through to devising an entirely new set of components for an SOA.

This work investigates the above statement using Web Services as the SOA implementation and found that, to a degree, the above statement is true. The only element of description required for a web service is the Web Service Description Language (WSDL) document and this does indeed allow the detection of a small number of mismatches when represented using our minimal web service architectural style.

However from the literature we find that the above mismatches are only a subset of those that we argue should be detectable. In response to this we produce an enhanced web service architectural style containing properties and analysis supporting the detection of this more complete set of mismatches and demonstrate its effectiveness against a number of case studies.

Contents

List of Figures

List of Tables

Chapter 1

Motivation and Introduction

The practice of software construction in a component-based fashion heavily based on software components reuse has long been recognized as an important solution for the software crisis [McI69]. It is a powerful means of not only reducing software development costs in the long run, but also reducing the risk of project failure, improving software quality, shortening development time, and greatly increasing the productivity of the individual software developer [Som01, Gac98]. This vision is still fully to become a reality. Obstacles to date have ranged from various organisational to technical barriers. Technical barriers include the occurrence of architectural mismatches during systems' composition from various independent software parts.

An architectural mismatch [GAO95] occurs when two or more software components are connected to form a system, and those components make differing and incompatible assumptions about their interactions or the environment in which they exist. The presence of an architectural mismatch between the elements being composed within a system can hinder reuse in a variety of ways. Problems can range from preventing elements' composition altogether to experiencing undesired side effects at run-time. Hence, architectural mismatches must be handled appropriately [Sha95a], by either being avoided during development and/or system reconfiguration, or being tolerated at run time.

Approaches to the mismatch problem have been discussed previously in the existing literature. These include the use of formal models for mismatch detection both at design or composition time [Gac98, AA96, FS02] and at run-time [UY00], pattern and mediator based resolution of mismatch [KG98, DeL99, HGK+06, CN08] and avoidance through flexible packaging [DeL01].

The work in this thesis is most closely related to those works on composition time detection, utilising architectural styles to encapsulate the rules and properties required for mismatch detection. Architectural styles have much to offer in this respect: they provide a vocabulary of architectural elements; parameters for the architect to follow; and constraints to check the validity of the individually chosen attribute values, as well as the overall system configuration. For these reasons architectural styles are heavily employed in this thesis with a significant portion of the contribution taking this form.

Service-Oriented Architectures (SOAs) are becoming one of the main trends in the current engineering of software. Web services are a recent approach towards supporting SOAs, building from standards agreed upon by various community stakeholders, while avoiding proprietary middleware solutions. Put simply, a Web service is any system that provides a network interface that is described by a published Web Service Description Language (WSDL) [W3C06c][W3C06d] [W3C06e] [W3C06f] document and uses Simple Object Access Protocol (SOAP) [W3C06a] as its message format. In this respect it is fair to characterise Web services as being an integration middleware [Bak02] or standard for presenting the interface parts of SOA [FS05] [Beh03]. Hence, using web services, as defined by W3C Web Services Architecture Working Group [W3C06b], supports the engineering of SOAs by providing rules and restrictions that apply to the definition of web services and how they can interact with other components to form a larger system.

1.1 Thesis and Goals

Given that SOA components are expected to make descriptions of themselves available, then provided a suitable description of client components are also available it should be the case that it is possible to detect architectural mismatches when bringing these components together to form a system. It is precisely this area that this work will explore, ultimately aiming to answer the following questions:

Central Questions. *Is the stipulated description of Web Service components sufficient to allow detection of all relevant architectural mismatches? If not, then what properties should both the services and the clients that use them make explicit to allow all relevant mismatches to be discovered? Finally, are architectural styles a suitable approach to support the representation and analysis of Web Service compositions for mismatch discovery?*

While there are many description documents that can be associated with a web service component [Pap08] they are only required to provide a WSDL description of themselves. An examination of WSDL in comparison to the architectural characteristics presented in the literature [Gac98, BJPW, DeL99, DGP02a] indicates that it does not contain sufficient coverage of properties to allow mismatch detection. From this the first thesis for this work to test is derived:

Thesis 1. *It is not possible to detect, at configuration time, all architectural mismatches in a system comprising of web services given only the minimal web service description and specifications.*

This question naturally leads to a more positive second thesis to test:

Thesis 2. *It is possible to describe a set of characteristics and rules that would allow all architectural mismatches relevant to web service components to be detected at configuration time.*

A number of smaller questions can be used to guide the work towards testing the two main theses. The first of these directly relates to the first thesis question and sets a baseline for the following work.

Sub Goal 1. *What mismatches could be detected currently?*

This only gives part of the information required to answer the first thesis, to complete the answer requires the following question also to be answered.

Sub Goal 2. *What mismatches are relevant in the scope of web services and their clients?*

The above goal highlights an aspect of the scope of this work. While the title mentions service oriented architecture, it will not be possible to detect mismatch unless the client components using the services are also represented. Thus client type components will also be included in the analysis styles.

Once the mismatches are known this will guide the development of an architectural style to allow their detection. The following two questions will also be answered in parallel as they are dependant upon each other.

Sub Goal 3. *What characteristics are required to represent the relevant mismatches and how can they be represented?*

Sub Goal 4. *What analysis is required to detect these mismatches?*

1.2 Assumptions of the Work

There are a number of assumptions that are made during this work, these and their motivations are listed below.

Dynamic Systems It is assumed that the software environment of SOA is dynamic with services appearing and disappearing as markets change and as they are upgraded. The result is that systems defined using this style must acknowledge that the components they are connected to may tear down the connections between them and cease to exist.

Organisational Separation 1 It is assumed that in a world of SOA the organisations developing client applications may be separate to the organisations developing services. Also a client may make use of services provided by more than one organisation. In this work then it is not possible for the developer of an application to know more about a component than is made available in it's public description. It is also only possible for a developer to make changes to their own component designs.

Organisational Separation 2 It is assumed that a web service may depend upon other web services to provide its functionality. For example a travel agent may offer flight availability information based upon several airline web services and a developer of a client for the travel agent service may not know about the existence or identity of the airline web services. The mismatch analysis therefore cannot assume it will have a complete view of the system.

Client Descriptions While it is currently true that web services should provide a standard (WSDL) description of their interface, the same is not true for client applications. Without some description of the client application it would not be possible to detect mismatches, so this work assumes that developers will produce description documents of their client designs.

Other description documents While a system design process may involve many documents describing the requirements of the system including the goals of any stakeholders, this work does not assume they are available for the purpose of mismatch detection.

Ontologies This work assumes that ontologies exist covering a number of aspects of the work. Firstly for giving semantic descriptions of data items and secondly relating to the failure modes described by each port. Furthermore is is assumed that these ontologies are shared between organisations developing client, service and broker type components.

Exploration not Simulation It is not the intention of this work to attempt to simulate the interactions between web service components, but rather to explore the possible interactions between them. This means that the actual timing of messages and the specific values of data they may contain are ignored in favour of a more abstract model which considers message order and the semantics of the data only.

1.3 Structure of the Thesis

Chapter 2 provides some background, introducing software architecture, web services and summarising a number of key pieces of related work.

The contribution starts in Chapter 3 where the minimal web service architectural style is described. This sets the baseline for the work by showing what mismatches can be detected currently with a minimal service description.

Chapter 4 sees the work returning to the literature to ascertain what mismatches are considered significant for general software components and then explores which of these are applicable within the scope of web services.

Chapter 5 describes an enhanced web service architectural style that builds upon the minimal style to address the additional mismatches found in Chapter 4. This is where the data structures and associated analysis used to detect mismatches are defined.

Chapter 6 shows the evaluation of both the minimal and enhanced architectural styles using a number of case studies. The work then finishes with suggested future work in Chapter 7 and the conclusions in Chapter 8.

The main chapters are followed by nine appendices that are included to support the thesis document and for repeatability, but are not compulsory reading. Appendix A gives an introduction to ACME and ACME Studio, the architecture description language (ADL) and environment used throughout this work. Appendices B–E contain the complete ACME[Gro06a] descriptions of both architectural styles presented here and also the ACME descriptions of the main scenarios used to evaluate this work. Appendix F contains a description of the external analysis utilised by the enhanced style along with the complete Java source code for the plugins. Appendix G presents the tables used by the external analysis to determine if two message exchange patterns match and the final two appendices, H and I, give and introduction to the CSP constructs employed here and describe the templates required for the correct functionality of the style.

Chapter 2

Background

This work has two main focuses, Web Services and software architecture. The background starts with a description of both what Service Oriented Architecture (SOA) and Web Services are before touching upon some efforts at improving Web Service interoperability both through standardisation and more explicit description. The content then moves to look at software architecture in general before listing some of the many architecture description languages (ADLs) available and describing why the ACME ADL was chosen to support this work. The following section describes two aspects key to this work, architectural styles and a number of software architecture characteristics deemed significant for interoperability by the relevant literature. The final section discusses architecture mismatch itself along with a number of approaches to its avoidance and resolution in general components. This section concludes by touching on some works related to formal description and analysis of Web Service compositions and a statement describing what is "architectural" taken from the literature.

2.1 Web Services and SOA

This first section introduces SOA and Web Services to give an outline of the components and systems this work aims to detect mismatches in.

SOA is a term which can frequently be found in relation to web services, but the literature seems lacking in precise descriptions. This may be due to them being a paradigm and not a hard protocol, however the OASIS consortium has produced a reference model [OAS06] which outlines the key features of SOA along with their relationships. A direct quote from the model states:

> Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

The above statement along with the three key aspects of SOA cited by OASIS *(visibility, interaction and real world effects)* are used as the guidance in this work.

The use of web services is one of the possible ways to implement a SOA [Sta06]. Web services themselves have been the focus of many research papers, with attempts at characterising their behaviour [MMR06] and formalising their descriptions [Col04, YWD06, Yeu06]. These works concentrate on providing detailed formal models of specific narrow focussed aspects of web services and not the more broad architectural style presented later in this work.

The Web Services Architecture working group (WS-ARCH) of the World Wide Web Consortium (W3C) define a web service as follows [W3C06b]:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Papazoglou [Pap08] describes two distinct types of web service, programatic and interactive:

Programatic these are relatively simple informational services. They may take the form of a request-response pair such as requesting the current weather at a location, or may provide front-ends to complex business information systems. These functions are generally atomic in nature;

Interactive services are those where a function is delivered by composing multiple services into a single service that may require multiple message exchanges to complete and also may be stateful, where the service keeps track of client state between invocations, and transactional.

Whichever type of service is implemented, a key aspect of SOA discoverability is that clients are able to "use" the services, and this requires some kind of description.

2.1.1 Description and Interoperability

Having given an outline of what SOA and Web Services are, this section now presents both the basic description document of Web Services along with some of the efforts at standardisation and expanded descriptions.

The basic description of a Web Service component is contained within a Web Service Description Language (WSDL) document. This is an XML formatted document that contains six main element types [Top03]:

Import Web service descriptions can be spread over multiple files, this is where the additional file locations are defined;

Types definition of any non standard data types to be exchanged in messages, such as 'record' type data structures;

Messages defines the messages exchanged at the web service interface, where each message is named and can contain multiple data items;

Port type this describes each port in terms of the messages it exchanges and pattern employed[1];

Bindings each port can be bound to multiple concrete protocols, for example HTTP and SOAP; and

Services groups related ports together to represent a service.

In essence these documents describe the syntax and ordering of messages required by each port provided and required by a service, but no more. Beugnard *et al.* [BJPW] argue that while components can offer much to software engineering, if they do not behave as expected then it is either because they are faulty or they are being mis-used. They propose that components should be used with contracts at four levels :

Syntactic level interface definition language;

Behavioural level pre and post conditions;

Synchronisation level service object synchronisation, path expression; and

QOS (Quality of service) levels such issues as maximum/average response time, accuracy of the result or throughput. Issues at this level may be negotiated when the contract is set up between service provider and consumer.

WSDL only covers the syntactic level of this four level contract. It is not surprising then to find that there are a great many other WS-* description languages in existance today covering many of these aspects. For example, Papazoglou [Pap08] mentions 43 such languages. However, this list is not complete, for example Parastatidis *et al.* [PWW$^+$05, PW05c, PW05b, PW05a] have produced a number of web service descriptions that focus on SOAP as being the only allowed message protocol. These descriptions include some support for including Communicating Sequential Processes (CSP) descriptions of message choreographies and so reach higher than the four level scale described above. Similarly, Fiadeiro *et al.* [FLB06] have described a language, SRML, which provides primitives to describe service compositions and their message passing behaviour. One distinguishing feature of SRML is that it describes the expected behaviour of the composition in terms of properties the composition should adhere to rather than prescribing a choreography.

[1] These message exchange patterns are described in Chapter 3.

A description of these languages is not included as none of them are mandated by the W3C as 'required' for a web service; all are optional[2].

This great number of optional languages and the natural language nature of the W3C web service descriptions mean that there are many implementation details that are not well defined. The Web Services Interoperability Organisation (WS-I) has produced a number of 'profiles' detailing a great number of implementation details of web services in a mismatch avoidance effort. For example:

> XML 1.0 allows UTF-8 encoding to include a BOM; therefore, receivers of envelopes must be prepared to accept them. The BOM is mandatory for XML encoded as UTF-16.
>
> R4001 A RECEIVER MUST accept envelopes that include the Unicode Byte Order Mark (BOM)

The WS-I work also includes tool support to test service implementations against those requirements that are testable. These details are much closer to the implementation choices than the intended scope of this work suggests, but it is important to acknowledge that such efforts exist. Also while prescriptive specifications could remove mismatch by eliminating design choice, it is also fair to say that some freedom of choice is required to build suitable systems, to quote Shaw [Sha95a] commenting on the flawed idea of designing all systems using a single paradigm:

> Most fundamentally, different architectural styles have different strengths and weaknesses, and a system architecture should be chosen to fit the problem at hand.

2.2 Software Architecture

It would not be possible to examine architectural mismatch without considering software architecture. This section starts therefore with a general description of software architectures.

Software architectures represent the high-level design of a software system. They provide critical abstractions with which it is possible to reason about and describe the structure and behaviour of a system[3].

These then are abstract models of a software system, but for that model to have some kind of meaning the syntax and semantics that underlie that model must be defined. It is the purpose of architecture description languages (ADLs) to provide, to differing degrees, exactly this.

Unfortunately there is no consensus on the details of what should and should not be included in an architectural description. In their original work, Perry and Wolf [PW92], suggested that architecture consisted of elements, form and rationale, where each has the following meanings :

---

[2]It is possible that the characteristics described later in this thesis as being required for mismatch detection are actually made explicit in these optional descriptions. A study of this point would be of value but it was not possible to conduct it during this work.

[3]A more thorough introduction to software architectures may be found in such material as [BCK98, PW92, SG96].

Elements are the processing, data and connecting elements within the system;

Form weighted properties or choices, where the weighting indicates the importance of the property or the requirement to select among alternatives;

Rationale is the motivation for the various choices made in defining an architecture.

It was from this grounding that the so called "first generation" of ADLs were produced. Medvidovic and Taylor in 2000 [MT00] produced a classification framework which not only described what they, at that time, believed should exist in an ADL but also the key properties of the languages that fitted their characteristics. The top level of their classification criteria is as follows:

Components the unit of computation or a data store;

Connectors the building blocks which model the interactions among the components;

Architectural Configurations the connected graphs of components and connectors which form the architectural stucture;

Tool Support strictly not part of the language, but vital to perform analysis, assist with code generation etc.

Further to this the interfaces to both the component and connectors are often described:

Ports represent the interfaces provided and required by a software component;

Roles declare the endpoints of a connector, these attach to ports and in doing so form the configuration of the system.

2.2.1 Description Languages

Web Services, as already described, use WSDL to describe their basic interfaces; but it is the intention within this work to use an ADL to describe the components and their configuration and utilise the associated tool support to facilitate mismatch detection. The purpose of this section is to recount what the literature says should be included in an ADL, give a brief description of some ADLs and then finally to describe why ACME was selected to support the work.

Following their classification Medvidovic and Taylor described ten notations which matched the criteria and were considered ADLs. A significant finding of the study was the range of focus of the ADLS, from quite general, structural, relativly semantic free offerings such as ACME [Gro06a] through to domain oriented notations such as MetaH[BEJV96]. The languages also varied in their choice of formal underpinnings and the maturity of their tool support.

In 2007, Medvidovic *et al.* [MDT07] produced another study in this area, extending the criteria to be deemed an ADL even further. In this study they postulate that a software architecture is

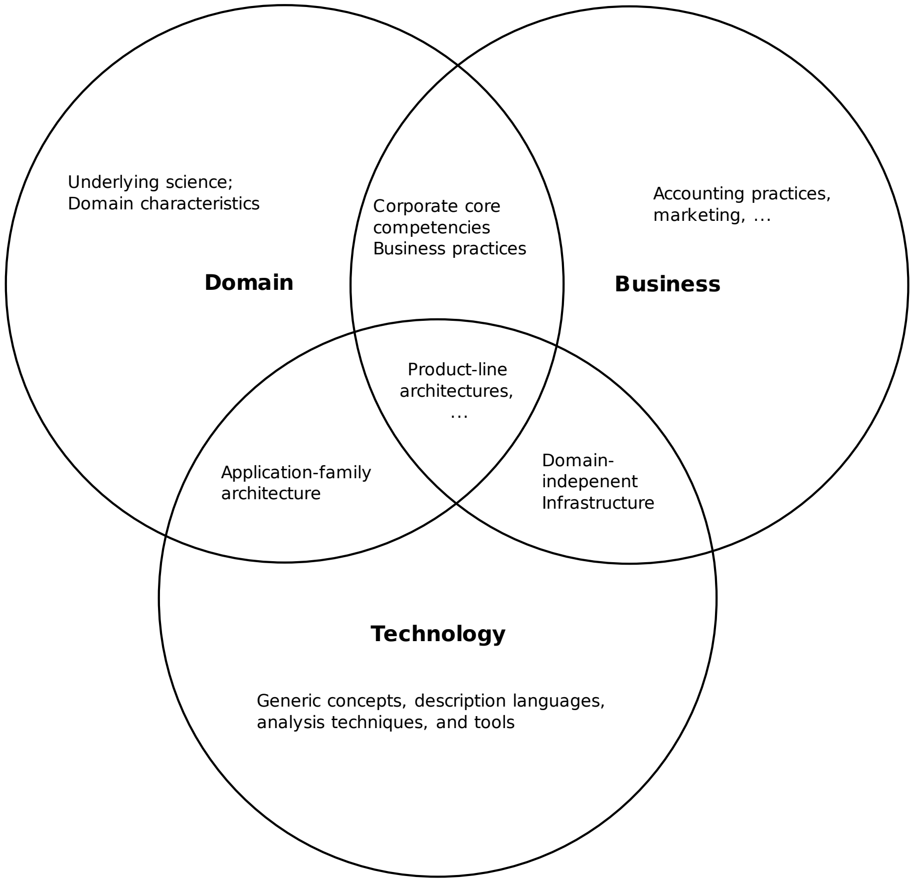


Figure 2.1: “Three lampposts” proposed by Medvidovic et al. [MDT07]

not simply a technological description of a system, but should include the viewpoints and requirements of other stakeholders involved in its inception. They propose that there exist three concerns which software architecture must address, technology, domain and business, but that the previous languages almost exclusively focus on the technology. They describe each area using a “lamppost” analogy, where each casts a light and there exist areas of overlap between them. Their Venn diagram representing this concept is reproduced in Figure 2.2.1

In their work, Medvidovic *et al.* argue that “second generation” ADLs should, as far as possible, provide support for all areas lit by the three lampposts; but, what they find is that there is no current notation that achieves this. Indeed, they do no expect that there ever will be a single notation that suits every project’s modelling needs due to the variety of domain and business specific requirements.

This work was initiated before the publication of the lampposts model, but it is interesting to look at where it fits in. Primarily this work’s view of architectural mismatch is a technological one, as was the case with the literature from which inspiration was drawn [GAO95, Gac98]. As such it does not come close to the accounting or marketing aspects which are given as examples on the lampposts diagram. At the same time the work does exist within the scope of the domain lamppost as it is focussed on web services and the definition of their “Domain characteristics” and the assumptions that can be made of them. In fact a large portion of this work is dedicated to the formalisation of these characteristics into an architectural style, which fits well into the “Application-family architecture” segment, which is described as addressing “technical problems that occur while building software systems within a target domain” [MDT07].

2.2.1.1 Summary of ADLs

A brief summary of ADLs taken from [MDT07], starting with the first generation ADLs is as follows:

ACME An interchange language for sharing architecture descriptions descriptions between tools, predominately at the structural level;

Aesop Specification of architectures in specific styles;

C2 Architectures of highly distributed, evolvable and dynamic systems;

Darwin Architectures of highly distributed systems whose dynamism is guided by strict formal underpinnings;

MetaH Architectures in the guidance, navigation and control domain;

Rapide Modelling and simulation of the dynamic behaviour described by an architecture;

SADL Formal refinement of architectures across levels of detail;

UniCon Glue code generation for interconnecting existing components using common interaction protocols;

Weaves Data flow architectures characterised by a high volume of data and real-time requirements on its processing;

Wright Modelling and analysis (specifically deadlock analysis) of the dynamic behaviour of concurrent systems;

The second generation ADLs as suggested by Medvidovic *et al.* can be summarised as follows:

UML 2.0 defines a set of views that can be used to represent a system or parts of a system. It is not specialized for modelling any particular domain and its diagrams and symbols do not have a formal semantics.

AADL is a language for specifying system architectures including both the software and hardware elements. It includes a number of predefined hardware and software types and these prescribe what kinds of properties may be specified about an element of a type. This language originates from MetaH.

Koala is an ADL derived from Darwin and is effectively a structural notation. It includes several constructs for supporting product line variability, such as switches describing variation points where a choice can be made about which implementation to use.

xADL 2.0 is an XML based ADL where types are described using XML schema. This allow users to add their own data types as needed by extending the existing schema, these schema can then be used to support syntactic checking of an xADL model. The existing xADL tool support focusses on the creation and manipulation of the XML schema and does not yet support the analysis a model's properties and structure.

2.2.1.2 Why ACME was selected

There were a number of unknowns at the point when ADLs were being considered. Firstly, it was not clear exactly what characteristics would need to be represented, and therefore an ADL that allowed flexibility regarding the properties included would be required. At the same time as not understanding what properties would be included it was not known how each would be represented, so an ADL that facilitated the inclusion of arbitrary data representations would be desirable.

A goal of this research was not only to enable the representation of the meta-data important to the detection of mismatches, but also to provide the rules which are employed to expose them. Architectural styles [SC97] (described later in this chapter) provide the means by which we can specify important characteristics and also the semantics and rules which apply to them. Style support was also then an essential element.

Finally, it was desired to be able to experiment with and test the outcomes of this study, so tool support which is capable of acting upon the constraints expressed in the architectural style to analyse system descriptions was also vital.

When the study commenced only one ADL stood out as fulfilling the above criteria, ACME [Gro06a], developed by Carnegie Mellon University. This language was developed as an architectural interchange language and so was designed from the outset to support the definition of arbitrary properties. ACME is also supported by a tool, ACME Studio [Gro06b] which offers a graphical interface and performs checks on an architectural description according to any ACME family (architectural style) it refers to. The rules are represented in a predicate language called Armani [Mon01], that allows the construction of boolean statements which are functions of the properties and existence of the architectural elements in the description. So ACME and ACME Studio provide a suitable environment in which to explore the representation of web services.

An introduction to the language and tool support of ACME and ACME Studio is presented in Appendix A.

2.2.2 Styles

As mentioned earlier, the ACME ADL and ACME Studio tool were selected partly for their support for architectural styles. This section gives an outline of what an architectural style is and provides references to a number of works that describe styles, one of which uses formally described styles to detect some of the architectural mismatches considered later in this work.

Architectural styles are a form of software design reuse [MKMG97, MG96]. At the simplest level they are used in name only, for example, stating that a system has a "client-server" architecture should give a mental picture of a single (or few) server components to which a larger number of client components connect to make use of their services. This is of course a very simple view but even so it can aid the forming of a mental model of the system in question, the roles of the components and even possibly hint at their behaviour.

Simply using styles by name can unfortunately be a source of misunderstanding as well. To quote Shaw and Clements [SC97]:

> After looking through the table many readers will say, "But that's not what I mean by style X!". Indeed, it may not be. But it is, as far as we can tell, what someone else means. This is an indication that different readers use style names in different ways.

Architectural styles however can offer much more than this general level of understanding, if used to their potential they provide the architect designing a system with three types of assistance. Firstly they can provide a **vocabulary** of elements which are expected to exist in a system of a particular style. Clearly in a *client-server* style system the components are either going to be clients or servers, but the vocabulary can also include the connectors, for example in a *pipe and filter* system the connectors between filters should be of the pipe type. The ports and roles of the components and connectors can be similarly named.

By themselves the names do not add much, but the second benefit of architectural styles, **properties** do. Each named type can have a distinct set of properties associated with it. The exact nature of these properties depends on the ADL, the environment in which it is used and the domain and purpose of the system being modelled, but they can range from primitive types such as integer values and strings to complex behavioural specifications and beyond. A server in a *client-server* system may for example have a maximum number of concurrent connections, which could be represented by an integer property, while the message passing behaviour it expects of a client could be described using a process algebra.

Finally, and in conjunction with the tools supporting it, the style can provide **constraints** and means for analysing a system. The constraints act upon the properties and configuration of the system and can tell us whether it is a valid instance of that style or not. The analysis can be used to model emergent behaviour of the system such as throughput or message passing conversations which

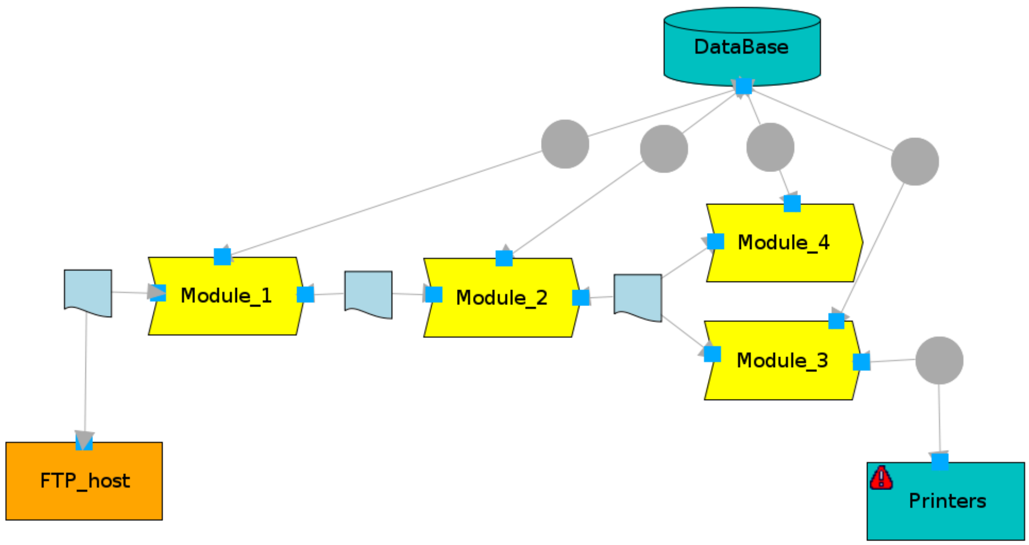


Figure 2.2: A simplified architectural view of a just-in-time delivery system. The two architectural styles present can be seen in the *pipe-and-filter* approach of the yellow main modules and the *shared-data* arrangement they take with the database.

can then be evaluated against system requirements or against component expectations for validity.

So architectural styles can guide an architect as to what elements should exist in a system of a certain type, prompt design decisions by providing properties which need populating with values and then offer feedback in the form of the constraint and analysis evaluations showing if and potentially where problems exist in a design.

The literature contains many references to styles, for example Gacek [Gac98] describes 11 styles including, *Pipe and filter*, *Black board* and *Event based*. Shaw [Sha95b] discusses seven styles, also including *Pipeline*, *Layered* and *Implicit invocation*. There are two points here, the first is that there are multiple styles already available that the architectures community value and the second is that they are often described in different ways and include different characterstics depending on their purpose– Gacek uses Z while Shaw uses natural language. This means there is no standard for how to represent a style or what characteristics should be included.

While many styles exist, it is not the case that all styles are appropriate for every system [Sha95b, SC97] or perhaps for a complete system. For this reason many systems exhibit characteristics of more than one style at the same time. For example, whilst working in indsutry the author experienced a just-in-time delivery system which could be described as both *pipe and filter* for the data processing to maintain the strict data ordering and *shared data* to maintain a consistent situational view (Figure 2.2).

The inclusion of multiple styles does not have to be at the gross level, it could be simply that the overall system style is *pipe and filter* but that the filters themselves are implemented using a *hierarchical* style.

While styles may not be mutually exclusive, this work reduces complexity by adopting a single style only view where all elements and configurations are expected to adhere to the web service styles presented. This imposes a view of the system including only the externally visible ports and abstracting away many internal details. This is argued to be the correct level of abstraction given that some of the web service components represented may belong to other administrative domains which may not be willing to share such internal details of their services or components.

2.2.2.1 Characteristics

The properties assigned to the architectural elements are at least as important to the final system as the structure itself. The issue is, what properties should an architecture description include? It is generally accepted that a model is an abstraction of a system that hides details not required for the purpose of that model. The same is true of software architecture, so the properties a particular description holds would be determined by its specific purpose and the analysis we might wish to carry out as can be seen in the dissimilar description methods applied to similar styles in the works of Gacek [Gac98] and Shaw [Sha95b].

There have been a number of works in which are described sets of characteristics that could be used in the description of architectures and styles. Those that proved influential in this work are now presented.

Shaw and Clements

In their *Field Guide to Boxology* [SC97] Shaw and Clements provide an early classification of styles using control and data as the dominant axis upon which to differentiate between styles. Their classification is divided into five features.

The first feature relates to the **constituent parts** allowed in a style, this is essentially the vocabulary as described in Section 2.2.2 and names the types of components and connectors allowed in a style.

Their second criterion **control issues** details the control flow between the components and the temporal properties they exhibit. This is broken down into three subcategories: **control topology**, describes the geometric form of the control flow graph in the system; **control synchronicity**, informs whether the control states of the components dependant on one another and **control binding time**, elucidates at what point in the component life cycle is the identity known of a partner in an exchange, *design time* to *invocation time.*

Data issues form the third category in their study and as with control issues this is broken down into multiple subcategories. **Data topology** refers to the geometric form of the data flow graph of a system. **Data continuity** describes the expected rates of flow of data through the system, this can range from *sporadic* to *continuous*, but also includes the notion of volume of data, ranging from high volume (*data intensive*) to low volume (*compute intensive*). **Data mode** indicates the means by which data is shared within a system, examples of which include *passed* in object oriented styles and *shared variables* in shared data systems. This also relates to the cardinality of elements receiving data in the exchange, *point to point* indicates a singular recipient while *multicast* indicates multiple recipients. **Data binding time** is the final data attribute that, as with the control binding time, describes when the partner in an exchange is known.

Control and data flows may not be independent of each other and so **control/data interactions** form the fourth set of characteristics. This is divided into two parts both referring to the relative geometry of the control and data flows. **Shape** indicates to what degree the shapes of the control and data flow graphs are isomorphic to each other. In the cases where these two graphs are similar, then **directionality** describes any relations between the directions of control and data flow, for example *same*, *opposite* or *none*.

Type of reasoning is the final category. Again this mirrors the analysis as mentioned in Section 2.2.2 and eludes to the analysis a style allows.

Gacek

In her PhD thesis Gacek [Gac98] describes tool support for describing the stylistic assumptions of components within a system and from that detecting architectural mismatches (described later in Section 2.2.3). The characteristics were given no explicit groupings and so are presented below, attempting to position related items closely.

Concurrency defines whether there is a constraint on the number of threads of control within a system. *Single-threaded* systems only have one thread of control passed between the components via calls while *multi-threaded* allow more than one thread to exist. Related to the concurrency property is that of **reentrance**. While a style may allow multi-threading, a component is only reentrant if the separate threads do not interfere with each other during execution. In the case that a style assumes multiple threads of execution it may also support the definition of **component priorities** which allow components performing more urgent roles to be executed in preference to others. In a similar area is **preemption** which describes the act of swapping out the current task on a processor and replacing it with another, some styles may allow this and others may not.

Distribution determines if a style constrains the mapping of processes to processor nodes. A *distributed* style either expects or allows the processes within it to exist on distributed hardware.

The **dynamism** property depicts whether the style allows for changes to its topology at runtime. This includes the creation and deletion of component and connector processes. **Reconfiguration** is the act of altering the topology or components in a system in some way, systems and styles may differ regarding whether this is allowed to happen on-line, off-line or at all.

A system which exhibits **encapsulation** provides a well defined interface to the components which use it, hiding other internal functionality. Similarly, the **layering** characteristic is used to represent whether there will be layers of components in the style, where each layer provides a virtual machine to the layer above it while using services of the layer below it. There should not be any bypassing of the layers above or below to reach more distant layers in this style.

Styles may also specify the **supported data transfers**, which are the means by which data is moved around, mirroring the data mode of Shaw and Clements. Styles may or may not have a **triggering capability**, that is some mechanism to allow the software to respond to events. State is also considered and the **backtracking** property determines if a component has the ability to return to an earlier state if required.

A **control unit** is a component which governs the execution of other components within a system. Some styles may require the presence of such a component.

The final characteristic is that of **response time**, which has three suggested values *predictable*, *bounded* or unpredictable.

DeLine

In his study of packaging mismatch DeLine [DeL99] proposed a number of assumptions a component may make about its environment and the components with which it will interact. In keeping with the previous works these characteristics are now summarised.

Components may disagree on the **data representation** they employ, which includes the type of data they are sharing (e.g. integer versus floating point) and the low level bitwise portayal of the value. In larger data structures such as a file containing a word processor document, the mismatching understanding of the representation may result in, for example, a loss of formatting.

Data and control transfer includes many aspects of the interaction between components. Firstly the mechanism by which the transfer takes place, for example a shared variable and also what is transferred during the interaction, data, control or both. Finally, whether the transmission is instigated by the sender or the receiver, usually termed *push* and *pull* respectivly. The number and direction of these transfers is captured by the **transfer protocol** property.

As with the backtracking characteristic of Gacek, state is considerd here. First is **state persistance** which considers whether state is maintained between interactions with a component and secondly **state scope** depicts the amount of its state a component allows others to affect.

The final two characteristics included in the work are **failure**, the degree to which a component will tolerate others' failures and **connection establishment** which is similar in nature to both control and data binding times of Shaw and Clements.

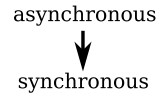


Figure 2.3: The ordering of concepts related to synchronisation from Yakimovich *et al.* [YBB99]

Yakimovich, Bieman and Basili

The study of Yakimovich *et al.* [YBB99] looked at a means for estimating the cost of integrating commercial off the shelf (COTS) software into systems. The basis for the costing estimates comparison of the various architectural asumptions made by the components and specifically those related to their interactions. The study identifies four major types of interactions: **Component–platform** between the component and the machine it runs on, e.g. assumptions about processor type. **Component–hardware**, the hardware devices the component interacts with, e.g. assumptions about the adresses of ports. **Component–user**, the interface provided to the user, e.g. assumptions about the language. Finally **component–software**, almost always components will interact with other components, e.g. assumptions about data representation.

Of these four the study focusses only on the component–software issues which are applicable to this work, these are divided into five subcategories. The approach taken for estimating the amount of "glue code" required is to compare the assumptions of the component to be integrated with those which make up the system and evaluate whether they are equal, compatible or incompatible. This evaluation is made possible by determining possible qualitative values for each subcategory and placing them into partially ordered sets, the ordering in these sets indicates compatibility, this is clarified in the description of the first category.

The **synchronisation** category captures whether a component blocks while waiting for a response from another. It has only two values, *synchronous* and *asynchronous* as shown in Figure 2.3. An asynchronous component could be made compatible with a synchronous one by including a loop to wait for a response to a call, so the arc in the diagram goes from asynchronous to synchronous to indicate an asynchronous assumption is potentialy compatible with a synchronous system. **Packaging** of a component represents how it is packaged for integration into a system, Figure 2.4. **Control** indicates the assumptions about the cardinality of threads and there location within the system, shown in Figure 2.5. **Information**, Figure 2.6, represents what is flowing between components in terms of data, control or a mixture of both. Finally Figure 2.7 depicts the types of **binding** expected in the system. The rationale behind these orderings can be found in the original paper.

Davis, Gamble and Payton

The above works, along with others, provide sets of characteristics that can be used to describe architectures and architectural styles, but they are not orthogonal and so an exercise in combination

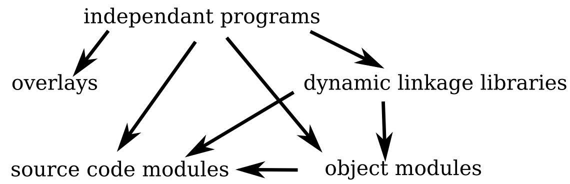


Figure 2.4: The ordering of concepts related to component packaging from Yakimovich *et al.* [YBB99]

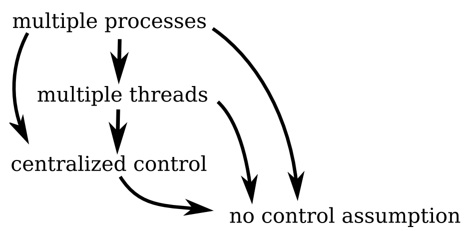


Figure 2.5: The ordering of concepts related to control from Yakimovich *et al.* [YBB99]

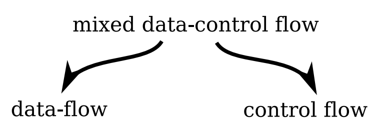


Figure 2.6: The ordering of concepts related to information flow from Yakimovich *et al.* [YBB99]

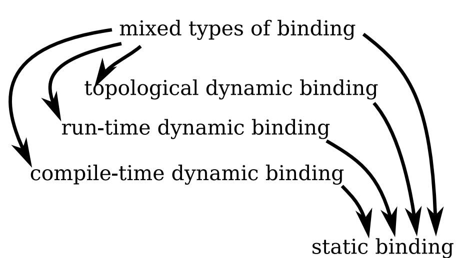


Figure 2.7: The ordering of concepts related to binding from Yakimovich *et al.* [YBB99]

Characteristic	*Values*	*Definition*
Identity of components	Aware, unaware	A components awareness of the existence or identity of those componenet's with which it communicates. Generally, filters in the pipe and filter architecture style are unaware, whereas object-oriented component names are used for method access.
Blocking	Blocking, non-blocking	Whether a component suspends execution to wait for communication. Most knowledge based systems run to completion without interruption and then wait, once done, for execution to be reinitiated.
Module	Filters, objects, layers, knowledge sources, blackboard data structures, control, interpretation engine, memory, process	Modules are loci of computation and state. Each module has an interface specification that defines its properties, which include the signatures and functionality of its resources together with global relationships, performance properties etc. The specific named entities visible in the interface of the module are its interface points.
Connector	Controller, pipes, procedure calls, shared data, implicit invocation	Connectors are the loci of relations among modules. Each connector has its protocol specification that defines its properties which include rules about the type of interfaces it is able to mediate for, assurances about the properties of the interaction, rules about the order in which things happen, and the commitments about the interaction

Table 2.1: The system classifications proposed by Davis *et al.* [DGP02a]

is required. Fortunately, Davis *et al.* have performed just such a task [DGP02a]. In their study the authors surveyed the available literature and found 74 separate characteristics, which by a process of combination and removing duplicates they reduced down to a set of 21 concepts.

We will now present their findings in similar form to the original work, before recounting the relationships between the characteristics which exist on three different abstraction levels.

The characteristics are divided into three groupings. **System characteristics** deal with the general coordination and characteristics of the style. This includes four characteristics, *Identity of components, blocking, module* and *connector* all of which are detailed in Table 2.1 where the description and suggested values from the paper can be found. The other two groups are **data characteristics** and **control characteristics**. As with the system characteristics, the descriptions of each are presented in Tables 2.2 and 2.3 respectively.

Reading the data and control tables it is apparent that the characteristics within each table are not orthogonal with some being refinements of others, this is a consequence of the three levels of abstraction employed by the study. The two semantic relationship diagrams shown in Figures 2.8

Characteristic	*Values*	*Definition*
Data topology	Hierarchical, star, arbitrary,	The geometric configuration of modules in a system corresponding to potential data exchange. A main/subroutine architectural style has a hierarchical data structure.
Data flow	No explicit values	The way in which data moves between the modules of a system. It clarifies the data interactions between internal modules and the exit points at which the data is made available. A pipe and filter style enforces a linear flow.
Data scope	Restricted, unrestricted	The extent to which modules internal to the component make their data available to other modules defines a component's data scope. In main/subroutine style a variable is only available for the subroutine in which it is defined and must be explicitly passed if needed by another function.
Method of data communication	Point-to-point, broadcast, multicast	Refers to how data is delivered to other modules. The method details whether data will enter a specific module at a specific point, e.g. pipe and filter architectures; if it will be delivered to those who have registered to receive it, e.g. event-based systems; or if it will be sent to everyone and only those who need it will use it, e.g. message queuing and broker systems.
Data binding time	Write, compile, invocation, run time	The time when a data interaction is established. A java process allows run time binding, making it possible to bind object classes together as they are defined.
Continuity	Sporadic, continuous	A general measure of the availability of data flow in the system. A pipeline has fresh data available at all times (continuous).
Supported data transfer	Explicit, implicit, shared	This delineates the type and format of data communication that a component supports as a precursor to actually choosing a method to communicate. For instance, implicit data transfer denotes an indirect mode of transfer as in an event-based system.
Data storage method	Repository, data with events, local data, global source, hidden and distributed	Details such as what type of data and how in the system will it be represented are gleaned from the chosen value of this characteristic. A blackboard architecture pattern utilizes a repository, namely the balckboard. Knowledge sources both store and retrieve data in this common space so that they may share knowledge.
Data mode	Passed, shared, multicast, broadcast	How data is communicated/transferred, in the logical sense, throughout the component. An event-based architecture will often broadcast its data.

Table 2.2: The data classifications proposed by Davis *et al.* [DGP02a]

Characteristic	*Values*	*Definition*
Control topology	Hierarchical, star, arbitrary, linear, fixed	The geometric configuration of components in a system corresponding to potential data exchange. A main/-subroutine architectural style has a hierarchical control topology.
Control flow	No explicit values	The way in which control moves between the modules of a system. It clarifies the control interactions between the internal modules and the exit points at which the control is made available. For example, control flow is bidirectional between modules in a hierarchical topology.
Control scope	Restricted, non-restriced	The extent to which the modules internal to the component make their control available to other modules defines a component's control scope. In a main/subroutine style, certain modules are scoped to receive control only from a parent module.
Method of control communication	Point-to-point, broadcast, multicast	Refers to how control is passed to other modules. The method details whether control will enter a specific module at a specific point, e.g. pipe and filter architectures; if it will be delivered to those that have registered to receive it, e.g. event-based systems; or if it will be sent to all and only those that need it will use it, e.g. message queuing and broker systems.
Control binding time	Write, compile, invocation, run time	The time when a data interaction is established. Unix pipes and filters bind at invocation time.
Synchronicity	Lockstep, asynchronous, synchronous, opportunistic	The level of dependancy of a module on other modules control state. It can operate either when no one else has control (synchronous) or during the execution of other components (asynchronous). Decentralised components are most often asynchronous. On the other hand, a main/subroutine style has synchronous control.
Control structure	Single-thread, multi-thread, decentralised	A measure of both the state of control and the possibility of concurrent execution. Control can reside solly with one module (single-thread), it can reside in multiple modules (multi-thread), and it can reside in multiple modules without any knowledge of other execution states (decentralised). A web-based interface will often have a decentralised control structure, whereas a pipe and filter style will utilise only a single thread.
Concurrency	Multi-threaded, single-threaded	The possibility that modules of a component can have simultaneous control. The number of threads present in the component denotes the concurrency. Databases support interleaved concurrency in transaction processing to allow multiple users to access a single account.

Table 2.3: The control classifications proposed by Davis *et. al.* [DGP02a]

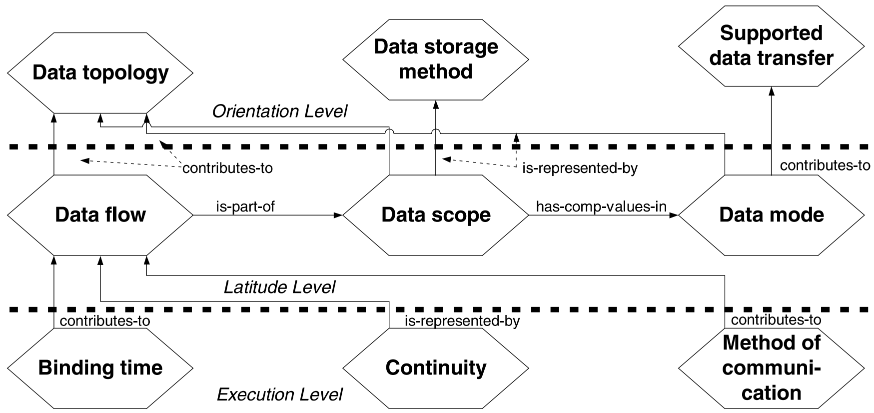


Figure 2.8: The semantic relationships and abstraction level of the data characteristics proposed by Davis *et al.* [DGP02a].

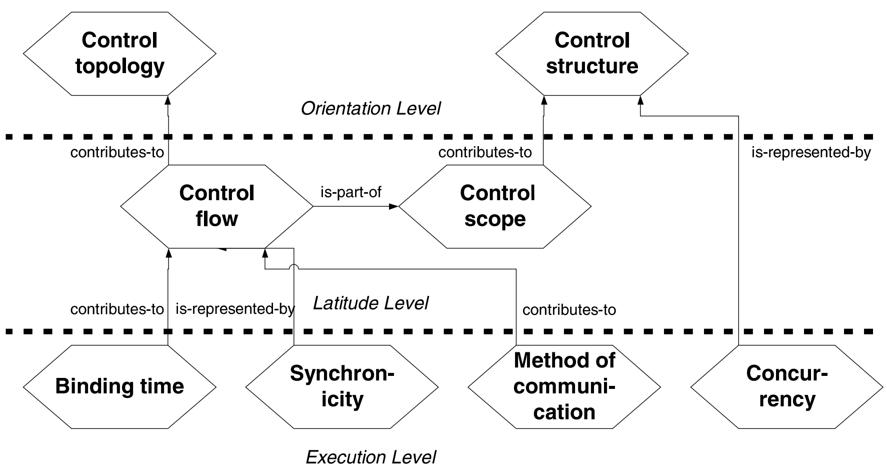


Figure 2.9: The semantic relationships and abstraction level of the control characteristics proposed by Davis *et al.* [DGP02a].

and 2.9 show the semantic relationships between and the abstraction levels of the characteristics in the data and control tables respectively.

The three abstraction levels are:

orientation level the most coarse grained, relating to the application requirements and the components within it;

latitude level this is finer grained than the orientation level and represents the *where* and *how* data and control flow through system; and

execution level the lowest level, it provides details such as data structures and other implementation details.

The four relationship types are:

is-a-part-of X *is-part-of* Y if and only if X and Y are at the same abstraction level and either X has attributes embodied in Y or X performs functions also used by Y;

has-comparable-values-in X *has-comparable-values-in* Y if and only if X is at the same abstraction level as Y and there exists at least one value in X that can be mapped onto at least one value in Y;

contributes-to X *contributes-to* Y if and only if X is at a lower level of abstraction than Y and X extends or refines some part of Y;

is-represented-by X *is-represented-by* Y if and only if X is at a lower level of abstraction than Y and the functionality of the value of X is reflected in some way by the value of Y.

2.2.3 Mismatch

The final aspect that requires introduction is the class of fault that inspired this investigation. This section starts by outlining what an architectural mismatch is before citing three examples. It then continues by discussing two sets of related work. The first is the literature presenting methods for either avoiding or resolving mismatch in general software components, while the second cites a number of works that employ formal descriptions of Web Service components for the purpose of detecting certain types of mismatch. The section concludes by recounting, from the literature, a possible definition of what "architectural" actually means.

Architectural mismatches prevent the successful integration of components to form a system. Architectural mismatches were first discussed by Garlan et al., when they introduced the term [GAO95]. To quote:

> Architectural mismatch stems from mismatched assumptions a reusable part makes about the structures of the system it is to be part of.

In their paper Garlan *et al.* describe a number of problems they encountered during the construction from component parts of Aesop which is, ironically, a platform to experiment with architecture development environments. The paper includes both the actual problems encountered and the groupings Garlan *et al.* derived from them. The problem groups are presented below with a description of the actual mismatches encountered in the following section.

Nature of components This category includes assumptions about the substrate on which the component is built (infrastructure), about which components will control the computation sequencing (control model) and about the way the environment will manipulate the data managed by a component (data model).

Nature of connectors This category contains assumptions about the patterns of interaction characterised by a connector (protocols) and about the kind of data communicated (data model).

Global architectural structure This category includes assumptions about the topology of the system communications and about the presence or absence of particular components and connectors.

Construction process In many cases the components and connectors are produced by instantiating a generic building block. For example, a database is instantiated, in part, by providing a schema; an event-broadcast mechanism is instantiated, in part, by providing a set of events and registrations. In such cases these building blocks frequently make assumptions about the order in which pieces are instantiated and combined in a system.

Key conclusions from their experience are a number of recommendations to support the construction of systems from components, these are now summarised:

Make architectural assumptions explicit A key problem is that the assumptions made during development of a component are not documented. This is not just part of the general problem of lacking documentation but also exists because there is no convention for documenting the architectural assumptions that the paper discusses;

Use orthogonal subcomponents The architectural assumptions of a system can be spread out among the components it comprises, this makes altering the configuration more difficult than just changing the links between components;

Provide bridging techniques In the paper several components are reverse engineered to overcome mismatches, this can be very costly. Bridging techniques such as mediating connectors and wrappers can help reduce these costs[4];

Develop sources of design guidance If sufficient intuition regarding which patterns of components work well together is not available then designers may use trial and error. The software community must find ways to codify and disseminate principles and rules for software composition.

The following three subsections outline motivating examples of mismatches. The first two present situations where the conflicting assumptions have been discovered after the system has been composed, in one case leading to the costly failure of an interstellar mission. The final example shows that while the problem of mismatching assumptions is known, there are still tools being used in industry that do not verify a system is free of even some of the simpler mismatches discussed in this work.

[4] An example of a mediating connector appears in the car parking scenario seen in Chapter 6 of this work.

Aesop

Garlan *et al.* encountered a number of difficulties during the development of Aesop, theses were attributed to the assumptions made during the development of the components it was built from. A selection of these mismatches will be outlined now.

The first example relates to the data structures owned by part of the graphical user interface (GUI), Unidraw. It had a hierarchical model that assumed that access to any child object would be through the top level parent object. While the data in Aesop was indeed hierarchical, the hierarchical access approach did not match with the intended use of Aesop which required that child objects could be modified directly. The resolution here was to create a flat data structure in Unidraw and implement a parallel hierarchical structure to represent the dependencies between parent and child objects.

Also related to data assumptions are the differing approaches taken by the Softbench event broadcast and the Mach remote procedure call (RPC) mechanisms used to facilitate inter-tool communications. Softbench assumed that most communications would be about files and their contents and represented data as ASCII strings, while Mach assumed it would be connecting components written as C programs and so used C data structures. In this case extra interfaces were implemented to perform translations between the incompatible data structures used.

The Aesop project expected to make use of two types of tool interaction, notification and request/reply. Softbench handled both mechanisms using the same callback structure for all three massage types. This meant that to implement a request/reply interaction a tool required two call back routines, one for the first message and one for the response. The result of this was that if a tool that had already sent a request, was itself sent a request or a notification then the callback routine associated with it would be invoked, forcing the tool to handle multiple threads and concurrency even if this was not a natural choice for the tool in question.

The final mismatch of significance to this work relates to the topological assumptions made by the OBST database utilised. It assumed a data centric star topology, with the database at the centre and no interactions between the surrounding tools at all. This caused problems when tools cooperating in a some action attempted to release the database to each other and forced the implementation of a transaction manager to hide these interactions from the database.

The overall result of the discovered mismatches was that the first Aesop prototype was achieved after 2.5 man years of effort rather than the 0.5 - 1 that was originally estimated.

Mars Climate Orbiter

A noteable example of a costly failure due to mismatching assumptions is the failed NASA Mars Climate Orbiter mission. Johnson [Joh05] tells us that the probe utilised an asymmetric solar array rather than a symmetric one, this necessitated the inclusion of a flywheel to counteract the small torque the array imposed on the probe. However as the flywheel velocity increased it became a threat to the safety of the mission and so had to be desaturated of kinetic energy by braking, this braking force was then countered by a firing of rocket motors. The mismatching assumption was in the sementics of the values being used to calculate the thrust required from each burn, with one software component assuming metric units and another using imperial units. The mishap investigation report [NAS99] cites the root cause as follows:

> The MCO MIB [Mars Climate Orbiter Mishap Investigation Board] has determined that the root cause for the loss of the MCO spacecraft was the failure to use metric units in the coding of a ground software file, "Small Forces," used in trajectory models. Specifically, thruster performance data in English units instead of metric units was used in the software application code titled SM_FORCES (small forces). A file called Angular Momentum Desaturation (AMD) contained the output data from the SM_FORCES software. The data in the AMD file was required to be in metric units per existing software interface documentation, and the trajectory modelers assumed the data was provided in metric units per the requirements.

This mismatch could have been detected if the architectural assumptions of all components involved had been explicit as suggested by Garlan.

Industrial Tool Allowing Mismatches

Even after the intervening years there still exist design environments that allow the construction of systems containing mismatches that go undetected and unreported. One such tool, which cannot be named because of commercial sensitivities, allows, for example a connector to be defined between component ports where none of the ports expects to write data onto the connector. Such a connector would serve no purpose and the attached ports would not receive any data, this is unlikely to be desirable and should be flagged to the designer.

2.2.3.1 Avoidance and Resolution

Since the phrase was coined a number of interesting works have been produced relating to architectural mismatch, the focus of these can generally be divided into two groups:

Mismatch Avoidance: includes means for either reducing the number of options available so mismatch is not possible or tools and techniques for detecting mismatch when it exists; and

Mismatch Resolution: techniques and patterns for handling a mismatch once it has been detected.

Gacek [Gac98] and Abd-Allah [AA96] both use the formal language Z to define architectural styles and systems and also to detect a selection a mismatches between the components.

Fukuzawa snd Saeki [FS02] use a similar approach, except in their case they use the coloured Petri Net formalism to assess if there is mismatch between the composed system and its specifications in terms such as reliability, resource efficiency and security. In this case the authors make the following admission:

> It may be difficult for practitioners and untrained persons to describe software architectures formally with CPNs [Coloured Petri Nets].

It is possible that this applies to any system that requires the user to construct a formal model before analysis can be performed.

The detection approach of Uchitel and Yankelevich [UY00] is to augment an existing system architecture model with additional labelled transition systems (LTS). These assumption LTS do not contribute additional behaviours to the system but instead restrict it as required by the assumptions they represent, for example, indicating the number of invocations of a service before old data must be purged to maintain performance. The LTS can then be monitored at run-time to detect mismatches in such non functional properties.

DeLine's approach [DeL01] falls into the mismatch avoidance category. He advocates that the early binding of functionality to a packaging method gives reduced flexibility. Instead he proposes that separating the functionality from the packaging and then building the packaging when the target system is known would increase flexibility. If a "ware" came with a high level specification of its channels and the target system had a specification of its required packaging then a packager component could generate "glue code" to produce a component that is directly integrateable with the target system.

In his earlier work DeLine [DeL99] follows a mismatch resolution approach. In it DeLine describes a number of abstract patterns that may be employed to mediate between components that mismatch on a number of characteristics.

Keshav and Gamble [KG98] also adopt the pattern based approach to resolving mismatches describing a number of patterns based upon combinations of three component types:

Translators change the data in some way;

Controllers control the communcations between components; and

Extenders which add functionality.

Cavalaro and Di Nitto [CN08] describe a framework called SCENE that allows a client application to connect to semantically equivelant services that differ in the details of their interfaces, for example in the number of messages exchanged to complete identical transactions. An example from this work is used later in Chapter 6 as part of the evaluation of the styles developed in this thesis.

2.2.3.2 Web Services Composition

A number of works exist that closely relate to aspects of this thesis in that they explicitly consider the composition of web service components.

The majority of these works describe the use of a formal language to both describe and in some way analyse the composition of components in terms of the messages passed. A variety of languages have been employed including Extended Finite-state automata/Promela [Nak06, Nak05], Petri-nets [VvdA05] , Coloured Petri-nets [YTX05] and Message Sequence Charts [FUMK03]. These approaches allow for analysis such as deadlock freedom and reachability to be carried out on the composed systems.

A different approach is presented by Ait-Sadoune and Ait-Ameur [ASAA09]. In this case the authors describe tool support for generating Event-B[5] models from BPEL [JE07] documents. The Event-B models are then passed into the RODIN[6] tool. This tool generates proof obligations resulting from the model and can discharge a number of them automatically. So while this proof approach does not suffer from the same state space explosion problems, it could require a user skilled in the Event-B formalism if the proof obligations cannot be automatically discharged or if some functional aspects of the services are to be verified.

These approaches are related to part of the work presented in Chapter 5 although the work here differs in a number of ways. The enhanced style presented uses a different formalism, CSP [Hoa85], to detect different mismatches relating to assumptions about the concurrency support of specific components along with both unexpected and missing messages, though the latter two could be linked to the deadlocks mentioned earlier.

2.2.3.3 Architectural Scope

While the above hints at what architectural mismatch is and how it may be tackled, it does not form a definition, certainly it begs the question, what is in and not in the scope "architectural"? Eden and Kazman [EK03] describe two orthogonal criteria that can be used to differentiate between three strata of specifications, *architecture, design* and *implementation*. The two criteria are:

Intensional/Extensional specification *"a specification is **intensional** iff there are infinitely-many possible instances thereof. Conversely, all other expressions are **extensional**."* Another way of expressing this is that a specification is intentional if it can be satisfied by an unbounded number of programs.

[5]Event B, `http://www.event-b.org/`
[6]RODIN, `http://rodin.cs.ncl.ac.uk/`

Local/non Local specifiction the authors quote from Monroe *et al.* [MKMG97] *"Architectural designs are typically concerned with the entire system"*. They go on to state that the difference between architectural and design specificiations is that *"architectural specifications must be met by every extension of the program"*, this suggests that design specifications are local, i.e. need only be satisfied in some part of the system. Another way of describing a local specification is that it can be satisfied in "some corner" of a program without being affected by what the rest of the program is like.

These criteria define the three strata of specification as follows:

Architectural specifications are *intensional* and *non-local*

Design specifications are *intensional* but *local*; and

Implementation specifications are *extensional* and *local*.

Eden and Kazman use architectural styles to demonstrate what intentional and non-local mean in less abstract terms. As an example they describe two rules relating to the layered architectural style described by Garlan and Shaw [GS93]. The first rule states that each element in the system is defined in exactly one layer while the second rule is that each element may only depend on elements in the same layer or in any lower layer.

They argue that this specification is intentional because it is obvious that an unbounded number of programs may meet this specification, due to there being no constraints on the nature of the elements in the system other than their dependencies. Furthermore they argue that the specification is non local as it may be violated by any component in the system depending on another that exists in a higher layer.

The ideas of intentional versus extensional specification and local versus non-local scope will be used during the final analysis of this work to evaluate the nature of the properties and analysis performed, with the purpose of justifying the use of the term "architectural mismatch".

Component versus System properties

The previous section describes the guidance that will be used to determine if a characteristic can be considered architectural. This section describes a second criteria that will be applied to determine if a characteristic should be included in the style. Figure 2.10 illustrates the two different types of assumptions that may be made. First there are the assumptions made by the components within the system about the properties of the other components that they will interact with. Secondly there are the assumptions made by the architect about the properties of the resulting system, this last set will be termed goals.

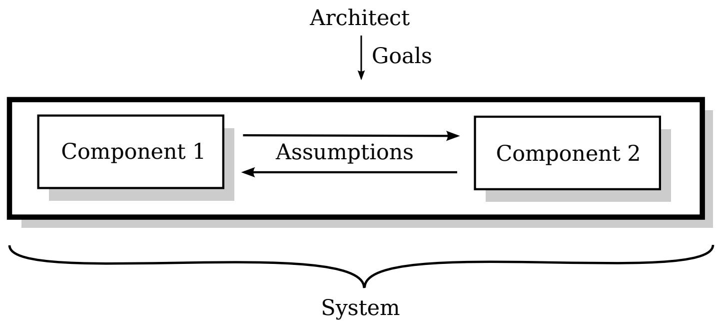


Figure 2.10: Components 1 and 2 make assumptions about how they will interact, these assumptions are within scope for this work. The architect may define goals for the system, such as job throughput, these are out of scope for this work.

The significant difference between the two is where the desired value for a property would be described. A component-to-component mismatch exists if one component makes a different assumption to another about how they are going to interact. The assumptions about how a component expects to interact will be known to the designers of those components and therefore they could be included in the description of those components.

A system property-to-goal mismatch occurs when a property does not meet the goal of the architect. In this case the system property may not be completely determined by a single component and may be an emergent property of the system as a whole and so would require some analytical technique to predict its value. An example of such a property would be the throughput of a system which depends on the throughputs of the individual components and their configuration. While it is not hard to imagine how the throughput of a system is computed, in the more general case the definition and compositionality of non-functional properties is not well understood [PF10]. Of greater significance to this work is the issue of where the desired values for these properties would be expressed. These goals are defined by the architect, not by the designers of the individual components and so the desired values would be expressed in an architect's goals document, which would be separate to the component descriptions.

While it is the goal of this work to determine what mismatches can be detected when composing a system from web service components, it is not the goal to either solve the issues associated with composing non-functional properties or to develop the additional description documents that would be required. For these reasons, characteristics that would be expressed in this architect's goals document will not be included in the style.

2.3 Summary

This chapter has introduced the main concepts and items of work upon which the following thesis is based.

There are works described in similar areas of mismatch but those aimed at mismatch detection relied heavily upon a knowledge of formal methods while those biased towards avoidance used pattern based approaches. The work that follows presents an architectural style based approach to composition time detection that employs predefined templates to reduce the formal methods expertise required for use.

The works also hinted that there are a great many architectural characteristics that the WSDL document does not contain. As WSDL is the only description document mandated for a web service to provide this means there are likely to be architectural mismatches that are not guaranteed to be explicit during composition of a system. It is interesting then to consider what mismatches are definitely detectable in comparison to those that the literature suggests are significant.

The contribution of this thesis begins in the next chapter where a minimal architectural style representing the properties of WSDL and the mismatches they can cause is described. The following chapters then build upon this to include many of the other characteristics from the literature, resulting in a style that detects a more complete set of mismatches.

Chapter 3

Minimal Web Service Architectural Style

The purpose of this chapter, the first of those in which we detail our contribution, is to determine the base line upon which the remainder of the work is grounded.

The literature around architectural styles provides us with a number of characteristics that could be considered when building a style. The base line consists of two parts, a list of those characteristics that could cause mismatch and are guaranteed to be made explicit, and a list of characteristics that are not guaranteed to be included in the description of a web service component. The first half of the chapter is dedicated to discussing these characteristics, the criteria for placing a characteristic on the first list and finally presenting both lists.

The second part of the chapter is devoted to developing the minimal architectural style based upon the characteristics in the first list. Here we show how each of the characteristics can be represented using the native data types in ACME and also that it is possible to construct rules in the associated predicate language, Armani, that detect all the mismatches associated with the minimal style. The characteristics in the second list are considered in more detail in Chapters 4 and 5.

3.1 What is a Minimal Web Service?

To be able to analyse web services for the purpose of building an architectural style we first need two things:

- A set of characteristics that an architectural style might contain; and
- A description of web services from which the values to populate the characteristics may be drawn.

For the first item we turn to three of the main works referenced in Chapter 2, specifically these are the outputs of Shaw and Clements [SC96], DeLine [DeL99] and Gacek [Gac98][1].

For the second part we desired a description of 'standard' web services to work from; we found this provided by the W3C working group on web services architectures [W3C06b]. This group defines a web service as follows:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

This description, along with the W3C descriptions of WSDL [W3C06c, W3C06e] and SOAP [W3C06a] forms the basis of our model of what a minimal web service is, and more importantly, what assumptions can be made about them.

In the following section we present characteristics from the literature that, in the light of the W3C descriptions, we found to be relevant to our style. To be relevant, a characteristic must meet two criteria

- Its effects are visible outside the component; and
- It may adopt more than a single value.

The first criterion stems from our aim of detecting mismatches that exist between components in a system and so if the effect of a choice is not visible externally then it cannot cause a mismatch within our scope.

The second criteria aims to remove redundant data if it cannot contribute to a mismatch. For example, web services encode their SOAP messages using XML. This is certainly visible externally but cannot be the cause of a mismatch since all web services will do the same. However while all web services use SOAP, the W3C currently hold descriptions of both SOAP 1.1 and SOAP 1.2. From this we can imagine that if a web service client expecting to use SOAP 1.2 attempts to interact with a web service using SOAP 1.1 then there is at least the possibility of interoperability problems and so this should be flagged as a potential mismatch.

We now move on to present the set of characteristics we found to be relevant to a minimal web service. A complete list of all the characteristics can be found in our technical report describing an early version of the minimal architectural style [Gam07].

[1] Davis *et al* [DGP02b] is not included in this list as the paper was not discovered until after the minimal architectural style was complete and the work had moved on to developing the enhanced style. The real value of their work was in guiding the characteristics to be considered for the enhanced style (Chapter 4) and it would not have changed the contents of the minimal style itself and so this chapter was not altered to include it.

3.1.1 Characteristics Relevant to the Web Services Based Architectural Style

Only two of the topological characteristics found have any bearing on the architectural style we produced, the first of which was **infrastructure and resource availability**. This characteristic captures the dependency assumptions a component makes about the system, such as the interfaces it expects to find in the supporting software and hardware infrastructure [Gac98]. While we found no constraints on the geometry of web service system topologies, it is fair to assume that a web service consumer will only attempt to connect to a web service provider interface.

Also under the topology banner comes **connection establishment**, which covers two aspects: when is the identification of a component, with which a connection is made, known, and how is the identification made available to the component. For both aspects there are differences between components that consume a provided service (the client) and those that provide it (the service). There is an underlying principle in SOA that services should be discoverable, which in turn implies that prior to an interaction neither the service nor the client know each other's identity. This strongly points towards components in a web service architecture not being pre-bound in any way. The second aspect also differs between clients and services. Clients are supposed to discover services and therefore their identification, a URI, by searching registries and then using binding information held in a WSDL document. Services on the other hand will likely only discover the identity of the client when the interaction starts through some mechanism in either the transport protocol or the message packaging as clients are not obliged to publish any interface description before using a service.

From the Characterisation category [Gam07] we found several more relevant properties. The first two items, **components** and **connectors** [SC97], are a broad statement about what types of components and connectors we expect to find in a system. In software architectures that are based on the use of web services it is valuable to distinguish between three different types of components given the specific roles that they play. These are: *services* that are web service components available to be discovered and integrated in various applications; *clients* that require services available as web services; and *intermediaries* that act as mediators between the clients and various services. Note that clients and intermediaries may be web services themselves, and there may be any number of intermediaries mediating interactions between clients and services. Given that web services are an implementation of SOA [Sta06], we deduce that they must provide access to some logical resource via a networked interface. Also from the W3C[2] we find that to be considered a web service the component must have an interface described by a WSDL document and also utilise SOAP as its message format. The associated connectors are largely unconstrained except that clearly they must

[2] http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211

carry SOAP messages and be compatible with whichever transport protocol is used by the web services.

Data mode [DeL99] refers to the abstract mechanism employed by a component to share data, such as a shared memory location, a broadcast message or an explicit transfer in a method call. Along with the choice of mechanism, it also includes the concepts of pass by value or by reference. SOAP messages are sent on a point to point basis between component ports and, as they are the only allowed means of communication in the style, it follows that the data they contain is passed on a by-value basis.

Data representation [DeL99] refers to the syntactic manifestation of the data being shared between components. At its simplest level this could mean the bitwise representation of an integer, for example how many bits long it is and if it is big endian or little endian. With larger data structures, such as a spreadsheet document, the components also need to agree on details of the structure in which the data resides. For web services both of these issues are resolved by the use of SOAP, which gives both a commonly understood structure and set of primitive data types that may be used.

None of the characteristics that fell within internal behaviour were constrained by either web services or SOA descriptions, so we move on to the external behaviour characteristics.

Here we found that the characteristics of **data and control transfer** and **transfer protocol** [DeL99] were both greatly influenced by the web service specifications. The two characteristics refer to components agreeing on what is transferred during an interaction, data and/or control and on the number and direction of transfers. These, with the possible exception of control transfer which is still implicit, are very clearly encapsulated in the message exchange patterns defined for web services, which are described next. Though these patterns only describe individual client or service ports, they do not extend to the longer term choreography between them, for example the fact that a component may expect an interaction on port 1 before it will allow an interaction on port 2 is not included.

There are two distinct versions of web services description language (WSDL), the main description language used by web services, WSDL 1.1 [W3C06c] and WSDL 2.0 [W3C06f]. These languages allow designers to describe the interfaces provided and required by a web service. The two versions perform largely the same function, but they differ in one main respect: WSDL 2.0 offers an extended set of message exchange patterns compared to those in WSDL 1.1, these will now be briefly described.

The **out-only/in-only** message exchange pattern, called **notification/one-way** in WSDL 1.1 terms, consists of just a single message sent from one port to another with no response. This is shown in Figure 3.1.

The **robust-out-only/robust-in-only** pattern, which has no equivalent in WSDL 1.1, extends

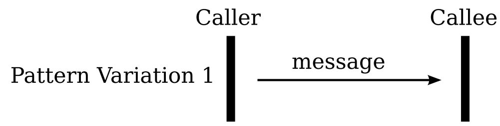


Figure 3.1: WSDL 1.1 Notify/One way and also WSDL 2.0 Out-only/In-only that exhibit the same message exchange pattern

the previous pattern by allowing an optional message in response which would indicate a fault has occurred. This is shown in Figure 3.2.

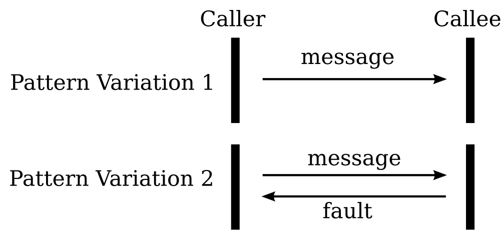


Figure 3.2: WSDL 2.0 Robust-out-only/Robust-in-only message exchange pattern

WSDL also allows for two way message patterns, **out-in/in-out** called **solicit-response/request-response** in WSDL 1.1, is the first of these. It consists of a single message sent from one port to the other which is then expected to reply with either the correct response or a message indicating a fault. This is shown in Figure 3.3.

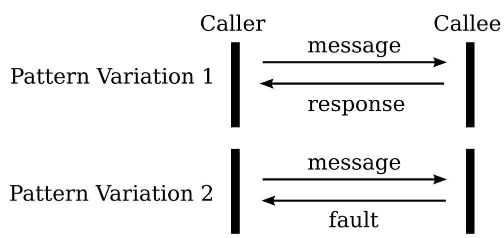


Figure 3.3: WSDL 1.1 Solicit response/Request response and WSDL 2.0 Out-in / In-out message exchange pattern

The final message pattern included is **out-optional-in/in-optional-out** and has no equivalent in WSDL 1.1. This pattern starts with a single message sent from one port to the other that then has the options of replying with the correct response, sending a fault message or not responding at all. In the case that it sends the response message the port that sent the initial message can then send a fault message if necessary. This pattern is shown in Figure 3.4.

The above patterns are presented in their matching pairs, there are also a number of pattern pairs that could be described as partial matches. A partial match is where the message patterns expected by one port are a proper subset of the other's. In this situation it may be possible to constrain the behaviour of the port with the super set of message patterns such that it behaves in accordance with

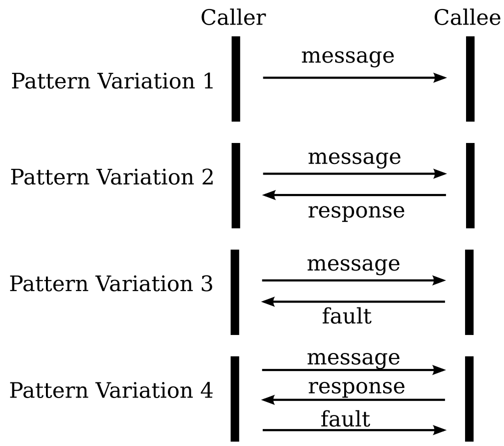


Figure 3.4: WSDL 2.0 Out-optional-in/In-optional-out message exchange pattern

the expectations of the other. An example of this would be a robust-out-only port connected with an in-only port (Figure 3.5), so long as the component with the robust-out-only port is prepared never to receive a fault then the two ports may interoperate. This is also true of a number of other message pattern pairs such as out-optional-in with robust-in-only and out-optional-in with in-out.

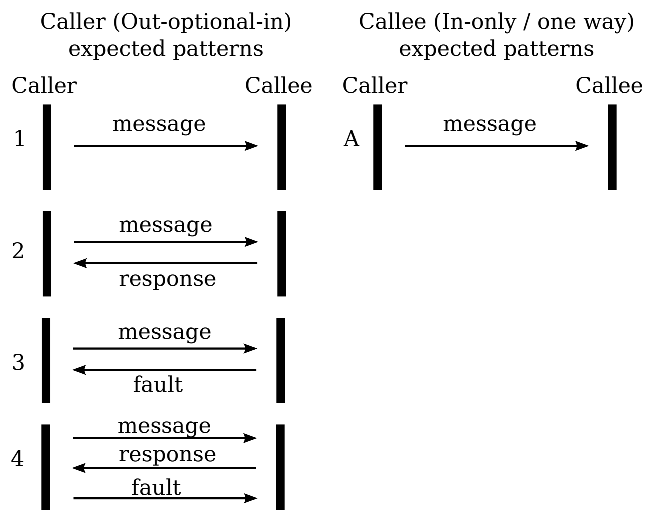


Figure 3.5: Partial match of Out-optional-in and In-only/One way. The callee message pattern “A” is only matched by the caller pattern “1”.

3.1.2 Characteristics Irrelevant to the Style Description

Many more characteristics are untouched by the minimal web service specifications, these then are not included in the architectural style we present.

In the topological field we find that neither the **data topology** nor the **control topology**, which describe the overall geometric form of the data and control flows, are prescribed by the W3C.

There is also no constraint on the **control/data shapes** or **control/data directions** that conveys if there are implications between the shape and direction of the control flows and the data flows, and vice versa. Thus the shape of the architecture of a web services based system cannot be characterised in the same way as say a pipe and filter system.

Internal behaviour is highly unconstrained, with characteristics like **state persistence** and **state scope** not described, meaning that components may or may not maintain state between invocations and they may or may not partition their internal state so the effects of one invocation are hidden from another concurrent invocation. Also, while there may be an intuition that a web service should have **concurrency support** in some way there is no constraint on if or how this is to be achieved.

There are several aspects of external behaviour that are not addressed by the standards either. **Control synchronicity** that looks at how dependent system components are upon each other's states is not touched upon. Dynamic properties such as the expected **data continuity** and **timing issues** are similarly untouched. Finally, while some message exchange patterns provide for fault messages to be sent as part of an exchange, **failure tolerance** and **error recovery** methods are neither constrained nor describable using the minimum set of specifications.

3.1.3 Summary

The findings of the above can be summarised into two lists. The first list includes those characteristics that are constrained or made explicit when complying with the minimum set of specifications applied to web services and the second includes those characteristics that are left free and at the choice of the designer of the component.

Constraints

- All components must be accessible to others via a network;
- each port on each component must be described by at least one WSDL document;
- each component must encode messages as SOAP;
- each connector must use transport protocols compatible with SOAP;
- service ports should allow clients to bind to them at invocation time; and
- data should be passed on a "by value" basis.

Freedoms

- Control topology is unconstrained and not made explicit in WSDL;
- control synchronicity is unconstrained and not made explicit in WSDL;

- data topology is unconstrained;
- data and control topologies are not constrained to be isomorphic;
- data and control directionality are not constrained by each other;
- data continuity is unconstrained;
- components may or may not maintain state;
- components may or may not support concurrent invocations;
- components are not constrained to respond in any timescale; and
- components may or may not support error recovery mechanisms.

3.2 Describing the Minimal Style in ACME & Armani

We now present the description of the style in its ACME & Armani form. This is comprised of the definition of the relevant ports, components, connectors, roles, and valid configuration rules. We first present the port types and data structures they use, followed by the component types then the single connector type. Finally we present the configuration rules. Note that there are no specialised roles in this style, so the default ACME roles with no explicit properties or rules are used.

3.2.1 Ports and Data Structures

The ports in this style contain all the properties required by the style. ACME supports inheritance between types so most of the properties are found in a `PortTWS_Common` type, with `PortTWS_Service` and `PortTWS_Client` extending and specialising from it, shown in Figure 3.6[3]. The definitions of the data types used by the properties can be found in Figure 3.7.

`PortTWS_Common` starts with an `EndPointList` property. End points are defined in WSDL and define the URI and message packaging protocol used by a port. A port may have more than one end point. This property, as with all those that do not have predefined values by the style, has an Armani rule to check that it is populated, which is considered to be requirement for a system to be compliant with the style.

Next in the `PortTWS_Common` definition we have the three properties that embody the message exchange pattern characteristics of a port. First we have `InOurControlDomain`, which determines if "we" have administrative control over a port, in which case it would be possible to alter its definition. This is vital to the rules defined in the connector that check compatibility of the message exchange

[3] All ACME & Armani descriptions presented here have had their comments removed for brevity of the descriptions. A complete description of the style including all relevant comments can be found in Appendix B.

patterns of two connected ports. This property uses a `SafeBoolean` type we defined due to not being able to confirm if a property using the native boolean type has been populated by the architect as ACME Studio assumes the default value of true if not populated.

The `MessageExchangePatterns` property represents the actual messages, their order and direction expected by this port. It is represented by a data type `messagePatterns`, which is a set of `validExchange`. A `validExchange` represents one complete path through the message exchange pattern as a sequence of `message`. Finally, a `message` is a record consisting of a string token representing the message name or syntax and a direction token that shows if the message is outbound or inbound from the point of view of the port that sends the first message. Thus we can completely describe the messaging behaviour expected by a port in a way that allows for message definitions to be refined as development continues.

In `PortTWS_Common` we also have `SendsFirstMessage`, a `SafeBoolean` type where we define whether a port sends the first message in the pattern or expects to receive it.

The definition of `PortTWS_Client` comes next. This port is identical to `PortTWS_Common` except that it declares itself in property `InInterface` to be part of the client interface. As previously discussed in Section 3.1.1 there is no requirement for client interfaces to be publicised so it needs no other properties.

Finally, we define `PortTWS_Service` that also extends `PortTWS_Common`. The service interface is required to be published so we have two additional properties here. `EndPointAddressList` stores a set of strings representing the address of that port. There are two rules associated with it, the first checks that the list is populated and the second checks there is one address for each end point offered by the port. The second property `WsdlDocRefs` is where the location of any WSDL documents that include this port is stored. This is not a functional property of the port, but, since it is required in SOAs that service ports be discoverable, this property has been included in the style.

3.2.2 Components

There are four types of component declared in the style, none of which have properties of their own but contain rules relating to the port types they can have, shown in Figure 3.8. The `CompTWSCommon` comes first and neither has any properties or rules, but it has been included as a place holder as future developments of this work may utilise it. The three types, `CompTWSClient`, `CompTWSService` and `CompTWSIntermediary` that extend `CompTWSCommon` all have a similar structure so are explained together. `CompTWSClient` represents a client component that only consumes services and thus its rules only allow it to have `PortTWSClient` type ports. `CompTWSService` represents a service provider and so its rules only allow it to have `PortTWSService` type ports. The third type `CompTWSIntermediary` represents a brokerage type component that offers services to some components while consuming services of others.

```
Port Type PortTWSCommon = {
    Property EndPointList : EndPoints;
    invariant size(EndPointList) > 0;

    Property InOurControlDomain : SafeBoolean;
    invariant InOurControlDomain == Yes OR InOurControlDomain == No;

    Property MessageExchangePatterns : messagePatterns;
    invariant size(MessageExchangePatterns) > 0;

    Property SendsFirstMessage : SafeBoolean;
    invariant SendsFirstMessage == Yes OR SendsFirstMessage == No;
}


Port Type PortTWSClient extends PortTWSCommon with {
    Property InInterface : Interfaces = Client;
}

Port Type PortTWSService extends PortTWSCommon with {
    Property InInterface : Interfaces = Service;

    Property EndPointAddressList : EndPointAddresses;
    invariant size(EndPointAddressList) > 0;
    invariant size(EndPointAddressList) == size(EndPointList);

    Property WsdlDocRefs : WsdlDocs;
    invariant size(WsdlDocRefs) > 0;
}
```

Figure 3.6: The ACME descriptions of the three port types defined in the style.

```
1     Property Type WsdlDocs = Set{string};
2
3     Property Type SafeBoolean = Enum { Yes, No };
4
5     Property Type legalSoapVersions = Enum { SOAP1_1, SOAP1_2 };
6     Property Type legalTransportProtocols = Enum { HTTP1_0, HTTP1_1 };
7     Property Type EndPoint = Record [
8         Transport : legalTransportProtocols;
9         Encoding : legalSoapVersions;
10    ];
11    Property Type EndPoints = Set{EndPoint};
12
13    Property Type EndPointAddresses = Set{string};
14
15    Property Type message = Record [
16        ST : string;
17        DT : string;
18    ];
19    Property Type validExchange = Sequence<message>;
20    Property Type messagePatterns = Set{validExchange};
21
22    Property Type Interfaces = Enum { Client, Service };
```

Figure 3.7: The data structures created to represent the properties used in the style. The `ST` & `DT` on lines 16 & 17 stand for 'syntax token' and 'direction token' in the message record type.

```
Component Type CompTWSCommon = {
}

Component Type CompTWSClient extends CompTWSCommon with {
    invariant Forall p : port in self.Ports |  satisfiesType(p, PortTWSClient) ;

    invariant size(self.ports) > 0;
}

Component Type CompTWSService extends CompTWSCommon with {
    invariant Forall p : port in self.Ports |  satisfiesType(p, PortTWSService);

    invariant size(self.ports) > 0;
}

Component Type CompTWSIntermediary extends CompTWSCommon with {
    invariant Forall p : port in self.Ports |  satisfiesType(p, PortTWSClient)
        OR satisfiesType(p, PortTWSService) ;

    invariant Exists p : port in self.Ports | satisfiesType(p, PortTWSClient) ;

    invariant Exists p : port in self.Ports | satisfiesType(p, PortTWSService) ;
}
```

Figure 3.8: The ACME description of the component types used in the style.

3.2.3 Connector

The style defines a single connector type `CompTWSCommon` that is shown split over Figures 3.9 and 3.10. The connector has no explicit properties of its own but it contains rules that make it the locus of mismatch detection. The first of these rules, shown in Figure 3.9 line 2, asserts that the connector may only have two roles, this is to embody web service connections being point to point in nature. The second rule, Figure 3.9 lines 4 - 8, is a check that two connected ports have end points that have at least one matching pair of end point protocols. The final two rules in Figure 3.9, on lines 10 - 12 and 14 - 15, check that one of the connected ports expects to send the first message and the other expects to receive the first message.

The final two rules, shown in Figure 3.10, are both concerned with checking the compatibility of the message exchange patterns of the two connected ports. The first rule is defined as a heuristic and the second is defined as an invariant, as are all the other rules in the style. This does not affect how they are evaluated but instead determines how a failure of a rule is displayed. When an invariant rule evaluates to false, a red warning triangle is displayed over the component or connector in question. However when a heuristic rule is failed then a yellow warning is given, indicating that a potentially less significant rule has been broken.

The message exchange pattern rules are based upon there being three possible outcomes of comparing the patterns of two connected ports. Remembering that a message exchange pattern is described using a set of valid exchanges, we define the first outcome, a complete match, as existing when the set of valid exchanges of one port is identical to that of the other. We can then also say that when the sets of valid exchanges are disjoint, we have a mismatch. However as we saw in Section 3.1.1 there are situations where one message exchange pattern may be a partial match for

```
Connector Type ConnTWS = {
  invariant size(self.roles) == 2;

  invariant Forall r1 : role in self.roles |
    Forall r2 : role in self.roles |
    Forall p1 : PortTWSCommon in r1.attachedPorts |
    Forall p2 : PortTWSCommon in r2.attachedPorts |
    (r1 != r2 AND attached(r1, p1) AND attached(r2, p2))
      -> size(intersection(p1.EndPointList , p2.EndPointList)) > 0;

  invariant Exists r : role in self.roles |
    Forall p : PortTWSCommon in r.attachedPorts |
    attached(r, p) -> p.SendsFirstMessage == Yes ;

  invariant Exists r : role in self.roles |
    Forall p : PortTWSCommon in r.attachedPorts |   attached(r, p)
      -> p.SendsFirstMessage == No  ;
    }
```

Figure 3.9: Part 1 of the ACME description of the single connector type defined, with the message exchange pattern rules removed.

another. We can now define two conditions for a partial match to exist, they are:

- one set of valid exchanges must be a proper subset of the other; and
- the port with the superset must be within "our" domain of control so "we" may reduce its set of valid exchanges to match that of the other port[4].

The two rules are constructed such that only one of them can fail on any one connector. So if the message exchange patterns completely match then neither rule will fail, if the conditions for a partial match are found then the heuristic rule will fail. Finally, if neither a complete match nor a partial match is found then the invariant will fail. In this way we are able to flag either a partial match or a mismatch being found and provide a visual clue to the architect regarding the degree of problem to be solved.

3.2.4 Configuration Rules

Finally we come to the rules that govern the configuration of the system. As we saw in Section 3.1.2, there are no constraints on the topology of a system of web services at all, but the web service style components will expect to connect to other web service style components. Also this style is aimed only at detecting mismatches between web services and may give false positives or negatives if other types of component are introduced. So two rules are defined, shown in Figure 3.11. The first states that all components found in a system of this style must satisfy the requirement to be of one of the three component types `CompTWSClient`, `CompTWSService` or `CompTWSIntermediary`. The second rule checks that all connectors in the system must satisfy the single connector type in the style `CompTWSCommon`, without which no mismatch detection will take place.

[4]The decision about whether this is possible or not must lie with the architect as understanding the implications of implementing the reduction in behaviour is outside the scope of this style.

```
  heuristic Forall r1 : role in self.roles |
    Forall r2 : role in self.roles |
    Forall p1 : PortTWSCommon in r1.attachedPorts |
    Forall p2 : PortTWSCommon in r2.attachedPorts |
    (r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
    (!
      (
        (p1.InOurControlDomain == Yes
         AND
         (!
           (isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns))
         )
         AND
         isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns)
        )
      OR
        (p2.InOurControlDomain == Yes
         AND
        (!
          (isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns))
        )
         AND
         isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns)
        )
      )
    );

  invariant Forall r1 : role in self.roles |
    Forall r2 : role in self.roles |
    Forall p1 : PortTWSCommon in r1.attachedPorts |
    Forall p2 : PortTWSCommon in r2.attachedPorts |
    (r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
    (p2.MessageExchangePatterns == p1.MessageExchangePatterns)
    OR
    (
      p1.InOurControlDomain == Yes
      AND
      (!
        (isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns))
      )
      AND
      (
        isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns)
      )
    )
    OR
    (
      p2.InOurControlDomain == Yes
      AND
      (!
        (isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns))
      )
      AND
      (
        isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns)
      )
    );
```

Figure 3.10: Part 2 of the ACME description of the single connector type defined, showing the rules relating to checking message exchange pattern compatibility.

```
Family ws_minimal_3 = {

  invariant Forall comp : component in self.Components | satisfiesType(comp, CompTWSClient)
      OR satisfiesType(comp, CompTWSService)
      OR satisfiesType(comp, CompTWSIntermediary);

  invariant Forall conn : connector in self.connectors | satisfiesType(conn, ConnTWS);
}
```

Figure 3.11: The ACME description of the configuration rules that check that all components and connectors in a system satisfy the requirements of this style.

3.3 Summary

In this chapter we have presented the derivation of our minimal architectural style. We started by discussing a set of characteristics that were obtained from the literature and met our criteria of being externally visible, non-trivial and explicit in a WSDL document. This resulted in a set of characteristics that are guaranteed to have descriptions and that can contribute to mismatch. This set formed the specification for the style, developed in ACME Studio that allows compositions of web services, clients and intermediaries to be assessed for a number of mismatches.

While the style shows us what mismatches we can detect, we also listed a number of characteristics that were considered important enough to be presented in the literature but that could not be determined from the description provided by a minimal web service. It is these characteristics that we will address in the following chapter where we will return to the literature to expand upon what they mean and if they are significant in terms of potentially contributing to mismatch in a system.

Chapter 4

Web Service Architectural Mismatches

Chapter 3 showed that with the expressiveness of ACME and the power of Armani it is possible to produce an architectural style that will represent the required data and provide analysis to detect the significant mismatches within the stated scope. Chapter 3 also focussed on a minimal web service, one that only makes available the compulsory set of data; but, even a cursory glance at the freedoms list in that chapter shows that there are still architectural mismatches possible that would go undetected by the rules in this style.

This chapter then has the goal of exploring the freedoms and determining what characteristics ought to be included in an enhanced web service architectural style and from that what mismatches the style is aiming to discover. The work now returns to the literature with the purpose of obtaining the details of these free characteristics and assessing their significance to SOA. Beyond simply listing the characteristics, values are suggested for representing the assumptions made by the components for each, without which it would not be possible to begin the task of designing the analysis rules to check for correct values and to detect mismatches.

Defining a scope is important for any work if it is to be successful. The scope here is to consider only those aspects that are common to web services and not be distracted by orthogonal characteristics. For example, Davis, Gamble and Payton included a **data storage method** characteristic in their work, this characteristic has suggested values including *repository* and *local data*. These are values that represent in some way the semantics of the component that are not related to it being a web service. The repository is described as being the main data store in a blackboard style system. However, while web services could be used to construct a blackboard system, they can be used to build other types of system as well. Because of this, the data storage method is counted as being out of scope of the web service style under construction and to check for architectural mismatches related to the blackboard characteristics of the system, that a blackboard architectural style should be constructed and used. This scoping issue also applies to a number of characteristics proposed by

Gacek [Gac98], these are detailed in the discussion of her work later in the chapter.

As discussed in Section 2.2.2.1 the survey by Davis *et al.* has performed a good portion of this work, but, as we will see, they do not completely cover all aspects of the other works described in the background. The chapter starts with the characteristics of Davis *et al.* but will then include characteristics from the other works by DeLine, Gacek and Yakimovich *et al.*, discussing if and where they overlap with the survey or what they add. When each characteristic is discussed, any envisaged mismatches relating to it will be named. At the end of the chapter a complete list of all named mismatches will appear to act as part of the specification for the enhanced architectural style to follow.

4.1 Davis, Gamble and Payton

4.1.1 System Characteristics

The first characteristic in the survey is **identity of components**. The survey proposed two potential values for it, *aware* and *unaware*. Web services send messages in a *point-to-point* manner [Pap08] and as such must be aware of the recipient's identity, at least in terms of its address. Also, the concept of broadcasting, which would be associated with being unaware of a message's recipients, is not associated with web services. So while a web service should always be aware of the identity of its partner in an exchange, it may be possible for the architect constructing the system to introduce a connector that suggests the use of multicasting by it having more than two roles. To protect against this and to enforce the point-to-point nature of web service communications the following mismatch is suggested.

mismatch 1: Non-point-to-point connector exists in the system.

The second characteristic is that of **blocking**, this is given the potential values *blocking* or *non-blocking*. While one might assume that web services, that exist in an open environment potentially without control over the clients that use them, would be implemented such that they can handle multiple requests concurrently, it is not actually stipulated by the W3C [W3C06b] that this should be the case. Therefore a web service component could adopt either a blocking or non-blocking approach. It follows then that this may result in a mismatch where a client assumes a non-blocking model while the service blocks.

mismatch 2: Concurrent calls to a blocking non-queuing port.

In the survey the characteristic of **module**[1] is given example values of *filter* and *object*, these indicate what type of component they are in terms of the vocabulary of the style. In this style the

[1] A module is a container of functionality and so is synonymous with a component. Thus the characteristics associated with modules in the survey are applied to components in this work.

actual role, in terms of functionality, is considered out of scope, it is assumed that a web service could perform any role. The only aspect of importance then is that a component is compliant with the constraints of being web service. If a component were not compliant in any way, this is considered to be a mismatch.

mismatch 3: Non web service compliant component in the system.

The final characteristic to appear in the system section of the survey is that of **connector**. Following on from the module characteristic this is given example values named after types of connector, for example *pipes*, *procedure calls* and *shared data*. As with the module characteristic, the style is not concerned with the details of the connector so long as it meets all the constraints, thus the existence of a non-compliant connector is considered to be a mismatch.

mismatch 4: Non web service compliant connector in the system.

4.1.2 Control Characteristics

The **control topology** characteristic refers to the geometric shape formed by the control transfers within a system and is given values such as "star" and "linear" by Davis *et al.* Terms like these have meaning when looking at the system as a whole but they do not apply to this web service architectural style for two reasons. Firstly there is no requirement that web service based systems form any particular shape in the specifications available [W3C06b]. Secondly when integrating a component into a system, the component is only directly affected by the behaviours visible to it, exhibited by its directly connected neighbours and the details of the control flows outside of this first ring of components is irrelevant. This then is similar to a layered architectural style, as described in [Gac98], in which components can connect to components in the same layer and the layers directly above and below them. The layer directly below a component effectively provides a virtual machine to it. Figure 4.1 shows an example of a web service composition in which layers exist, not in the virtual machine sense of the layered architectural style , as there is no prohibition regarding which components may communicate directly with any other, but in the sense of what can definitely be known. In the diagram the client component "knows" it is connected to components A1 – A4, however it may not know about the existence or connections to components B1 and B2 as components A1 – A3 may exist in a different administrative domain and may not wish to make that information available. As such there is nothing to gain by making assumptions about the topology beyond the directly connected components.

The focus now moves down to the latitude level (Figure 2.9, page 24) and to the **control flow** characteristic. Davis *et al.* did not suggest any specific values for this characteristic but did describe it as "[clarifying] the control interactions between the internal modules and the exit points at which

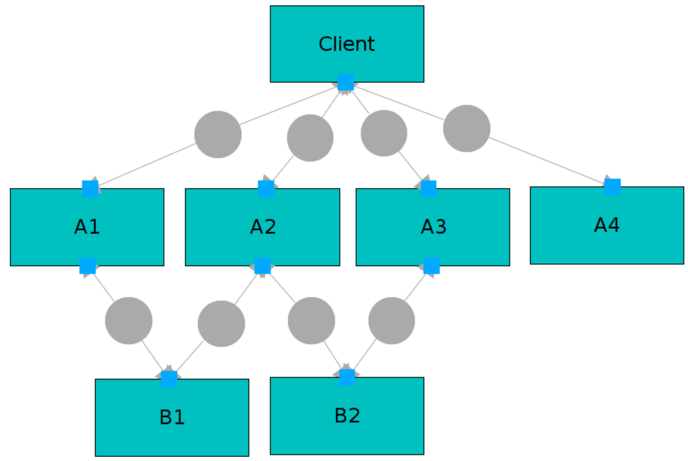


Figure 4.1: A client component that connects directly to components A1, A2, A3 and A4. Components B1 and B2 also exist in this composition but only the behaviour of A1–A4 seen by the client is significant.

it is made available". The only interactions visible at the abstraction level adopted by this style take the form of the messages passed between connected ports.

A mismatch at this level then is relatively easy to visualise in the form of a message sent to a port on a service when the service is not expecting it. Looking at the left hand side of Figure 2.9 (page 24) we see that control flow is fed by two execution level characteristics which are now discussed.

Control synchronicity describes the dependence of one component upon the control state of another, for example a synchronous component may not have control while another has it. Papazoglou [Pap08] observes that there are two distinct types of web services, synchronous services, using remote procedure call (RPC) type communications and asynchronous services using a document passing paradigm. From a control flow point of view these differ in that the former passes control to the service with the call and control is returned with the reply. While the latter passes the document but does not pass control or wait for an immediate reply. As presented in Section 3.1.1 there are several message exchange patterns that may be employed by web services and the communication paradigm chosen for each port is implicit in this choice. For example the *one-way* message exchange pattern, Figure 3.1 page 38, sends a message but does not expect any response and so is applicable in asynchronous situations. Conversely the *request-response* pattern, Figure 3.3 page 38, sends a single message then expects either a response or an error message before continuing and so is consistent with the synchronous paradigm. The mismatch then lies in the choice of message exchange pattern with the implied effects on the logical control flow of a component.

mismatch 5: Mismatching message exchange patterns.

Returning briefly to the control flow characteristic, note that the above mismatch does not capture the problem of an unexpected call to a port. For example, while two components may agree on using the asynchronous document passing paradigm, if they may not agree on the number or direction of documents passed, [DeL99], then there still exists a mismatch. A mismatch relating to the conversations each component expects is added in response to this.

mismatch 6: Mismatching conversational assumptions.

Control binding time is the other characteristic that feeds into control flow. This represents the point in a component's life cycle at which the identity of a partner in an exchange of data or control is first known. Web services, as an example of SOA, should be loosely coupled and as described in [Pap08] this means late binding. While this is certainly true of web service components that provide services, it does not necessarily apply to the client components that use those services, a client component may be bound at design time to use a specific service. This leads to a mismatch type being identified for web service components providing services, where they are bound to a set of client's components before runtime. In this case the mismatch does not cause any interoperability issues until a new client, that is not part of a predetermined set, attempts to make use of that service. The mismatch then is between the service provider and the general expectations that may be applied to a web service by a prospective client.

mismatch 7: Incorrect binding time of a service provider.

Control structure is given the potential values *single-threaded, multi-threaded* and *decentralised* by Davis *et al.*. From one point of view a system composed entirely of web service components must be decentralised as each component must have at least one thread of control to be able to either send a message to another service or to listen for incoming messages. On the other hand this could also be seen as a practical detail and if logical threads of control representing the value adding functionality are considered instead, then a different conclusion may be arrived at.

The goal of any system is surely to do something useful; Schneider [Sch00] defines the concept of '*liveness*' as "something good will happen". Schneider also tells us that a component on its own cannot guarantee that an event will occur given that the system may prevent the event by refusing to cooperate. In a system of web services, cooperation could be interpreted as a component being willing to send a message while another is willing to accept it.

Mismatches relating to the willingness to send or receive a message are already accounted for by mismatches 5 and 6, but for the system to do something useful then at least one of the components must start with a thread of control that will lead to it sending the first message. If none of the components have such a thread then no useful actions will take place and the system will not exhibit liveness.

The mismatch in this case may not be immediately intuitive as it is not an incompatibility between any pair of components, but instead results from all components in the system waiting for some other to send the first message. If we define a component that can send a message to another before it receives any messages as having an active thread of control then the following mismatch can be derived.

mismatch 8: No component starts with an active thread of control.

Control scope describes the restrictions a component places upon the other components that it is willing to share control with. While in the example given by Davis *et al.* a subroutine will only receive control from its parent, web services may receive a message and therefore a logical thread of control from any component to which they are correctly bound. No new mismatches are identified here.

The final two characteristics in the control section are method of communication and concurrency. **Method of communication** has the suggested values *peer-to-peer, multi-cast* and *broadcast.* As web services use the point-to-point (peer-to-peer) method, there should be no scope for mismatch but as with identity of components characteristic (Section 4.1.1) there could be a fault introduced if the architect introduced a non-point-to-point connector into the system. Mismatch 1, which was declared in that earlier section, is sufficient to cover this issue as well.

Concurrency has two proposed values, *multi-threaded* and *single-threaded.* As discussed in relation to the blocking characteristic previously, it may seem intuitive, given the open environment in which web services exist, that they *should* be logically multi-threaded and support multiple concurrent invocations, but this is not mandatory. The developer's choice of implementing technology affects this ability. For example web service constructed using Java servlets will support multiple threads [2] while those built using the Enterprise Java Beans (EJB) technology are strictly single-threaded with the number of threads supported by the component being then dictated by the number of beans instantiated [Top03]. In the cases where only a single logical thread can be processed it may be advantageous to include a queuing facility to store messages waiting to be processed. From this a single mismatch is derived.

mismatch 9: Concurrent threads attempted in a single-threaded component.

4.1.3 Data Characteristics

Data topology, similar to control topology in the previous section, represents the geometric shape formed by the data flows in the system. The same argument applies in this case as it did before, that a component may be unaware of the data flows details on the far sides of the components it directly interacts with. The component should therefore only make assumptions about the known

[2] Allowing multiple threads to exist does not strictly mean that they are supported as this would imply that logical concurrency issues such as race conditions are accounted for in the component development. These are discussed later when looking at the reentrance characteristic on page 59

data flows between it and the directly connected neighbours. No new mismatch types are identified here.

In another mirroring of the control characteristics, **data flow** has much in common with control flow. Again it represents the flows, this time of data, around the system. The mechanism used for implementing data flows is the same as that for control flows, i.e. the sending of SOAP messages, so we would argue that the previously described mismatches relating to control flows (mismatches 5 and 6) also apply here.

Data flow is fed into by all three execution level characteristics. **Data binding time** has the same SOA principles applying to it as control binding time had, i.e. that a service should be discoverable which implies binding at run-time. Also client components using the services have the same flexibility regarding binding time, i.e. at any point in the lifecycle. The need for mismatch 7, identifying a service pre-bound to specific clients still holds here.

Systems may differ with respect to their **data continuity**. This characteristic represents whether a component will always have fresh data available for it to process, such as an osciloscope, or whether the appearance of new data will be sporadic, such as a bar code reader at a supermarket checkout. These two situations are termed *continuous* and *sporadic* respectively. While web services do not stream data as such[3], there is no specification of an upper bound of the frequency with which messages are sent from one web service component to another. This could lead to an approximation of streaming if a sufficiently high rate of messages were sent and received. This effectively allows a mismatch in the continuity assumptions between services if one component expected only sporadic communications while the other expected a near continuous stream of messages.

mismatch 10: Differing data continuity assumptions.

The **method of communication** of data is the same for data as for control, i.e. point-to-point message passing only. So again, as there are no options no new mismatches are identified here. **Data Scope** also receives the same treatment as its control couterpart and a component will, if any security constraints are met, share data with any correctly connected component.

The **data mode** characteristic represents how a component expects to share data with others in the system. Davis, Gamble and Payton suggest a number of possible values, *passed, shared, multicast* and *broadcast*. As already discussed web services use a point-to-point messaging style, so multicast and broadcast are out of scope[4]. Shared data is not disallowed in any specifications found, but using shared data would imply that there exists one or more connectors in the system

[3] IBM has produced a plugin to allow a web service host to stream data such as video (`https://www.alphaworks.ibm.com/tech/streamingengine`), however no evidence was found that a body such as the W3C have standardised the way in which web services would handle stream requests, though a use case of streaming data was described in 2002 (`http://www.w3.org/TR/ws-desc-usecases/#N103D8`). (URLs correct on 5th November 2008)

[4] Multicasting is included in the WS-discovery specification (`http://xml.coverpages.org/ni2004-02-17-b.html`) but no other mention of it was found. Also the specification relates to methods for announcing and discovering web services rather than their operational message passing.

that represent the shared locations. One of the constraints imposed on the style is that only web service type connectors are allowed and these do not support shared variables, so shared data is not allowed in this style. This only leaves the passed option and thus no possibility of mismatch relating to this characteristic.

Data storage method has already been discussed in the introduction to this chapter, but to reiterate here, this characteristic is out of scope of this web service architectural style.

The final characteristic of Davis, Gamble and Payton is that of **supported data transfer** to which the authors assign two possible values: *explicit* or *implicit*. Web services are explicit about data transfers for two reasons. Firstly they use a point-to-point method, directly sending the SOAP message to its intended endpoint. Secondly they are explicit about the data included in those SOAP messages, at least in terms of each datum's name and data type, these are detailed in the web service WSDL document [W3C06c, W3C06e]. A client wishing to use a service does not have to publish a WSDL document, so its data names and type are not made explicit, but web services do not discover and bind to clients, the binding takes place the other way round. It is assumed that a developer of a client component has documentation regarding the interfaces required by that component, effectively making that data explicit, at least internally to those building the composition. Given these assumptions, the information regarding the data types to be used by a client application will always be *explicit* and so there is no new mismatch here.

4.2 DeLine

DeLine's work on packaging mismatch contains a number of characteristics that are positioned at the right level of abstraction to be of use in this study. While no example values are presented for any of the characteristics they are all illustrated with examples that help clarify the intent of each characteristic.

Data representation is the first characteristic presented. It has no direct counterpart in the Davis, Gamble and Payton study as it concerns how each data item is presented by a component. This presentation is in terms of the data type (floating point, integer etc) representing the value and also the bitwise representation of that value (most significant bit first or least significant bit first etc). Web services use the XML schema syntax to define the data types used in a WSDL service description document and therefore also by the component interface. SOAP itself also uses these data types so communicating web services will have a common understanding about the data types in use and their representation. This still leaves web services with a potential mismatch relating to the actual data types used in a particular exchange of messages. For example a stock broker service may quote prices in pence using an integer, while the client expects the price in pounds as a floating point number. Here we can see two distinct mismatches. The first is a mismatch of the actual data types exchanged. The second is the meaning of the values encoded, where one component's data could be described as "value in pence" while the second is "value in pounds". Both issues need to be set right for correct operation to occur.

mismatch 11: Mismatching data types in a message.

mismatch 12: Mismatching data semantics.

DeLine also states that with large data structures whether there is a mismatch or not is less of a black and white issue. This is illustrated with the example of a word processor document, while a word processor may be able to open a file of another vendor's product, some formatting information may be lost. The raises the possibility of a new mismatch type.

mismatch 13: Mismatching data structure.

Data and control transfer lies in much the same area as the Davis, Gamble and Payton control flow and data flow characteristics. DeLine breaks down a number of communication mechanisms using two criteria, *what* is transferred between the components and *who* requests the transfer. Web services always transfer data, whether control is passed or not is implicit in the message exchange pattern, so mismatches regarding what is transferred are captured already in mismatches 5 and 10. The issue of who requests the transfers is also already covered in the mismatches regarding the message exchange patterns and longer term conversations, mismatches 5 and 6 respectively. The same applies to DeLine's **transfer protocol** characteristic where the "number and order" of individual transfers would be described . This is precisely the purpose of the message exchange patterns and so mismatches with respect to it are already captured in mismatch 5.

DeLine proposes two characteristics relating to the state of the component. **State persistance** targets how much state is maintained between interactions of components. Papazoglou [Pap08] describes two types of web services, *informational services* that provide access to data such as weather reports, these do not keep any memory of the previous interactions and are considered stateless. The second type are termed *complex services*, these typically include multi-step business processes, for example purchasing, which could include requesting a quotation, placing a purchase order, confirming the order, delivery information and so on. This type of service must maintain state to be able to function. State can include the values of attributes of a component [ML05] and can also include the expected or allowed transitions of a component [CBB$^+$04]. Both of these could lead to a mismatch, the first in an assumption about whether a component's variables are stateful or not and the second in an assumption about the messages a service is prepared to send or receive. The former therefore leads to a new mismatch type while the second is included in mismatches 5 and 6. The significance of this characteristic is related to the concept of rely/guarantee [Jon81]. If component A assumes that state is maintained while component B assumes it is not and then B makes changes to some data, this could cause problems later if A relied upon the data having the earlier value.

mismatch 14: Mismatching assumption about statefulness of variables.

DeLine's second state related characteristic is **state scope**, this represents the assumption about the amount of scope a component is willing to allow another to affect. For example if a service allows multiple client applications to use it simultaneously then it may divide its internal state, allowing each client to affect only its own portion or it may share some state between the clients. The clients themselves may make assumptions about whether the state they interact with is shared or private, thus a mismatch is possible here.

mismatch 15: Mismatching assumptions about privacy of state.

Failure tolerance is the penultimate characteristic here. It represents the assumptions components make about the failure modes of others. DeLine gives the example of a component packaged to interact with a local hard disc, that instead receives its data over a network. The network may exhibit different failure behaviour to a local disc drive, possibly leading to the component making erroneous assumptions about the failure that occurred. A mismatch relating to differing failure modes is added to acknowledge this.

mismatch 16: Differing failure modes assumed and exhibited by interacting components.

Finally DeLine includes a **connection establishment** characteristic. This includes how and when the ID of a component to be interacted with is known. This has already been covered in discussion of the control and data binding times of Davis *et al.* that led to the inclusion of mismatch 7.

4.3 Gacek

Gacek's work on detecting architectural mismatch contains 14 characteristics, many of which were not covered by the scope of the Davis *et al.* study, so the complete set is presented below.

Concurrency was the only characteristic of Gacek's explicitly cited in the Davis *et al.* survey and has already been accounted for in mismatch 9.

The **distribution** characteristic describes assumptions about the mapping of processes to processor nodes. Problems may occur if a component expects its partners to exist on the same node but they are placed upon another due to the potential delays or errors caused by communications across the network. Web services, as their name implies, are primarily intended for service provision across networks, though this does not preclude co-locating web service components on a single node. There is the potential for mismatch then, though it is not clear what problems, other than performance related issues, would arise from this.

mismatch 17: Differing distribution assumptions.

Dynamism concerns assumptions about changes in topology at runtime. Certainly web services are oriented towards dynamic discovery at runtime using standard UDDI (Universal Description, Discovery and Integration)[5] registries. Papazoglou [Pap08] defines two types of web service clients, *static* that are pre-bound to a specific service provider and *dynamic* that understand the methods and parameters of a service type but do not bind to a particular service end-point until run-time. This means that the creation of a connector can certainly be dynamic, but topology changes can also include destruction of a connector to terminate a binding to a component. No detail was found regarding which ports involved in a connection are allowed to instigate the destruction of a connector. From this it is possible that the ports may differ in their assumptions about which of them may destroy the connection and thus leave the other waiting for a message that will not arrive or sending a message to a port that will not accept it. A new mismatch is added to represent the assumptions made about which parties involved in a connection may create and/or destroy a connector.

mismatch 18: Differing assumptions about who may create or destroy a connector.

The **encapsulation** of a component is considered next but this is not a source of mismatch for two reasons. Firstly encapsulation requires that a component has a well defined interface and web services are obliged to provide a WSDL document describing their public interface so this requirement is met. The other part of encapsulation regards whether the interface can be circumvented or not. In the minimal web service style presented in Chapter 3 we stipulated that all service ports must have an associated WSDL document, to reflect the first part of encapsulation. This acknowledges that the described service interface can only be circumvented if there are service ports that have no description. This rule will be maintained in the enhanced style that follows.

mismatch 19: Provision of an undescribed service port.

Layering in a style implies there are hierarchic levels in the topology of the system, each layer providing a virtual machine to the layer above and using the virtual machine below it. We previously discussed Similar aspects were previously discussed relating to Davis *et al.*'s control and data topology characteristics, concluding that there is no constraint on the geometric form of a web service system.

In a layered style components may only connect with components in the same layer or those directly above or below it. While the virtual machine metaphor from the layered style was used

[5] `http://www.w3schools.com/WSDL/wsdl_uddi.asp`

when discussing control topology earlier in the chapter (page 50), the associated rules about not bypassing the adjacent layers do not apply here and any suitable component may be connected to. The result is that no new mismatches are found here.

Supported data transfers is slightly different to the Davis *et al.* characteristic of the same name. While the latter is just concerned about whether the transfer method is explicit or implicit in nature, Gacek's version focusses on the type of mechanism used, e.g. *shared variables* or *data repository*. The mechanism used by web services is explicit message passing so there can be no mismatch in this respect.

The **reconfiguration** property characterises commitments about when a system can be reconfigured, either *on-line* or *off-line*. If a reconfiguration is taken to mean a change in topology then this equates to whether web services can bind and unbind on-line and/or off-line. We determined earlier in this chapter that web services should allow binding at run-time (on-line) and that client components can bind at any time, this gave rise to mismatch 7. Nothing has been found explicitly describing when connections between web services may be destroyed, also assumptions about this characteristic were covered when discussing dynamism, which spawned mismatch 18. For example a component making the assumption that reconfiguration can only occur off-line might assume that neither it or the component it is connected to may destroy the connector and so that connector will be available until the system is shut down. So no new mismatches are found here.

Reentrance is an important characteristic when considering systems with multiple threads of control. It describes whether a component supports multiple concurrent invocations of parts of its interface. The interpretation of supporting taken here relates to the component being protected against logical concurrency issues such as race conditions rather then the question of whether the framework upon which the web service is built actually permits two or more concurrent threads in the same component. The following mismatch is added to cover this.

mismatch 20: Concurrent threads in a non-reentrant method.

The final six characteristics in Gacek's work are all considered to be out of scope in the context of this web service style, we will now briefly describe why each is so.

Three of the characteristics **backtracking, control unit** and **triggering capability** all represent aspects of the semantics for use of a component in the same way as the data storage method of Davis, Gamble and Payton. Again whether a component has these characteristics or not is orthogonal to whether it is a correct web service or not, thus it is suggested that they would be better suited to existing in more application specific styles such as a blackboard architecture style.

The next two characteristics are related to how the operating system handles the components for execution. **Pre-emption** describes if a process may be "swapped out" so another process may get some processor time, and **component priorities** describe if defining priorities for each process within a system is allowed or expected. These two characteristics may affect how a web service component performs but not in a way that would affect the interoperability of the components.

Finally, the **response time** characteristic represents the degree to which the temporal aspects of an interaction can be predicted. This is an important aspect in the arena of quality of service (QoS), however service level agreements (SLA) are a subject in their own right and not an area targeted within this work. No mismatch will be included with respect to this characteristic.

4.4 Yakimovich, Bieman and Basili

The motivation for this piece of work was to aid in estimating the cost of integrating COTS components into a system, by assessing the degree of difference between the component to be integrated and the system using five characteristics. The characteristics are described below.

The **packaging** of a component describes the form in which it is to be found, where the form has values ranging from an *independant program* to a *source code module* with types such as *overlays* and *dynamic link libraries* in between. The context of this style gives three possible scenarios with respect to the packaging characteristic. The first is that all the web service components already exist and are immutable, in which case the role of the architect is to compose simply the system by creating connections between the desired ports. The second scenario is that a number of web service components already exist and the role of the architect is to design one or more new components to be integrated with the existing ones. The final role is that none of the components exist, or that they are all within the architect's control and therefore can be changed, so the role again is that of a designer. So while the context for the style does not allow for the full gamut of packaging, Yakimovich *et al.* propose it does allow the spirit of this characteristic to enter in the form of the mutability of the components and ports.

Mixing mutable and immutable components does not in itself lead to interoperability problems. This characteristic becomes important however when other mismatches are discovered as these may be corrected either through direct modification of the component or by using a technique such as those proposed by DeLine [DeL99] and Cavallaro and Di Nitto [CN08]. This drives the inclusion of a generic type of mismatch, the partial match. This does not target any particular property at this point but will be used in situations where a property of two components shares some commonality but is not completely compatible. An example from the minimal style would be a partially matching message exchange pattern where there exists at least one path that is shared by both components but there are also paths that they do not share.

mismatch 21: Partial characteristic mismatch between two or more components.

The **control** characteristic describes the sort of control flow expected in the system. Values here range from *multiple processes*, where each component has its own thread of control, to components such as a library which make *no control assumptions*. This concept was discussed in reference to Davis, Gamble and Payton's control structure where it was determined that all web services must have a thread of control to either listen for incoming messages or to initiate an outgoing message. It was decided that a component would only be considered to have a thread of control if it would initiate communications with another component without any external stimulation. This led to the formation of mismatch 8 "no component has an active thread of control" that applies equally here.

The **information flow** characteristic captures whether control, data or both flow between components during interactions. Both of these concepts have been seen before, with the control flow captured implicitly in mismatch number 5 and mismatches in data flow would be caught by mismatch 11.

Synchronicity only has two values, *synchronous* and *asynchronous*. Again this is a concept that has already been covered previously in the two mismatches relating to message flow expectations, mismatches 5 and 6.

The final characteristic we considered by Yakimovich, Bieman and Basili is that of **binding**, this evaluates the time at which the ID of the component to be connected to is known, once again this is already covered by a previously defined mismatch, in this case it is number 7, "incorrect binding time of a service provider".

4.5 Summary

In this chapter a number of sources from the literature have been reviewed for candidate characteristics for the enhanced architectural style. It was found that while there was a degree of overlap between the characteristics each includes, they were all able to add to the set of mismatches to consider.

There were also a number of characteristics that were found to be out of scope for the purposes of our work. For example the characteristics that relate to specific components such as the control unit proposed by Gacek or those that relate to the semantics of the component such as the data storage method proposed by Davis, Gamble and Payton. These characteristics are better suited to existing in application domain specific styles such as one that might describe a blackboard system. The characteristics that influenced the mismatches that will be used in the style are those that are oriented towards the interoperability and discovery of SOA.

This chapter has indicated that there are more mismatches suggested by the literature than are detectable using the WSDL description alone, the following tables will be used to illustrate this. The minimal style in Chapter 3 contained some 22 rules with their associated properties representing the features found to be significant, these are shown in Table 4.1 where each rule is assigned an ID of

ID	*Name*
	Port Rules – Figure 3.6 page 43
r1	Endpoint list must be populated
r2	Sends first message populated
r3	MEP populated
r4	In our control domain
r5	Service : end point address list populated
r6	Service : has address for each endpoint
r7	Service : is defined by a wsdl doc
	Component Rules – Figure 3.8 page 44
r8	Client : clients have only client type ports
r9	Client : has some ports
r10	Service : has only service ports
r11	Service: has some ports
r12	Intermediary : has only client or service ports
r13	Intermediary : has at least one client port
r14	Intermediary : has at least one service port
	Connector Rules – Figures 3.9 and 3.10 pages 45 and 46
r15	Has exactly two roles
r16	Attached ports must share a common transport and encoding protocol pair
r17	One attached port must send the first message
r18	One attached port must receive the first message
r19	Attached message exchange patterns should match
r20	Attached message exchange patterns may partialy match
	Configuration Rules – Figure 3.11 page 46
r21	All components must be web service client, service or intermediary
r22	All connectors must be web service connectors

Table 4.1: Rules specified in the minimal architectural style, Chapter 3

the form r<x>. While some of these rules directly relate to the detection of a specific mismatch, for example r19 confirms that the message exchange patterns match, several of the rules relate to the integrity of the architectural model, e.g. r6 confirms there is an address for each end-point provided by a service.

The underlying mismatch behind each of the rules is presented in Table 4.2 where each is assigned a minimal style ID (min<x>) and associated with the rules used to detect it.

This chapter has provides us with a second list of mismatches, those suggested by the literature. These are presented in Table 4.3, relating the mismatch ID (lit<x>) with the name of the mismatch. In this case the ID numbers are identical to those given in the body of the chapter.

From the tables then we can see that there are certainly more mismatches suggested by the literature than WSDL facilitates the detection of. In terms of overlap there are a number of instances where the minimal style mismatches concur with those from the literature, for example both lists contain references to mismatching/partially matching message exchange patterns. Table 5.1 on page 68 shows that there are 6 mismatches from the literature that the minimal style already

ID	*Name*	*Associated rule*
	Mismatches affecting interoperability	
min1	Message exchange patterns should match	r19
min2	Message exchange patterns may partially match	r20
min3	Connected ports have a common transport and encoding protocol pair	r16
min4	Message directionality should match	r17,r18
	Mismatches between elements and the style	
min5	Ports must be well defined	r1 – r7
min6	Components must have the correct port types	r8 – r14
min7	System may only contain web service compliant elements	r21, r22
min8	Connectors have exactly two roles, point-to-point	r15

Table 4.2: Mismatches checked by the minimal architectural style, Chapter 3

ID	*Name*
lit1	Non-point-to-point connector exists in the system.
lit2	Concurrent calls to a blocking non-queuing port
lit3	Non web service compliant component in the system
lit4	Non web service compliant connector in the system
lit5	Mismatching message exchange patterns
lit6	Mismatching conversational assumptions
lit7	Incorrect binding time of a service provider
lit8	No component has an active thread of control
lit9	Concurrent threads attempted in a single threaded component
lit10	Differing data continuity assumptions
lit11	Mismatching data types in a message
lit12	Mismatch of data semantics
lit13	Mismatch of data structure or syntax
lit14	Mismatching assumption about statefulness
lit15	Mismatching assumption about privacy of state
lit16	Differing failure modes assumed and exhibited by interacting components
lit17	Differing distribution assumptions
lit18	Differing assumptions about who may create or destroy a connector
lit19	Provision of an undescribed service port
lit20	Concurrent threads in a non-reentrant method
lit21	Partial characteristic mismatch between two or more components

Table 4.3: Mismatches determined during the literature review in Chapter 4

considers, this means that 15 of them are not detectable using the minimal style, and therefore WSDL, alone.

The literature does not only suggest a greater number of mismatches than the minimal style considers, but it also covers a wider scope. The minimal style only considers syntactic issues that are included in WSDL, such as the messages exchanged, the data types they contain and the transport/encoding protocol. The mismatches from the literature include similar concepts but also consider characteristics that go beyond a single pair of connected ports, such as the longer term conversations the components might expect, the semantics of the data a component exchanges and failure modes a component may exhibit to name but a few.

At this point it is interesting to reconsider whether the mismatches listed in both tables relate to web services or to SOA or to both. Certainly the mismatches presented in the minimal style are applicable in the web service domain, but do they also apply to SOA? The considered answer at this point is yes, the mismatches are applicable to both web services and to SOA in general. The caveat here is that while the mismatches do apply to both domains, the same does not apply to the rules used to detect those mismatches. A prime example of this is the mismatch *min 3 - Connected ports have a common transport and encoding protocol pair.* For two ports to communicate they must have compatible protocols, this is true for both web services and SOA, so the mismatch itself stands in both cases. The significant difference is that while web services are constrained to use HTTP and SOAP, the more general SOA paradigm does not prescribe any such constraint. For this reason the rules in the minimal style used to confirm that each port uses HTTP and SOAP of various version would not be suitable for SOA. A similar situation is found if we consider the message exchange protocol mismatch, this essentially stipulates that connected ports must agree on the number, direction and (syntactic) contents of the messages they exchange. This is surely as true for SOA in general as it is for web services, but again the difference would appear the rules used to check the correctness of the port descriptions with web services having to comply with the eight message exchange patterns defined for WSDL 2.0 while SOA is not constrained in this way.

If a similar view is taken of the additional mismatches suggested by the literature in this chapter then arguments analogous to those above can be found for all but a few of the mismatches. There are three specific mismatches that either do not apply or would require alteration to apply to SOA. The first of these is *lit 1 - Non-point-to-point connector exists in the system.* While web services are constrained to use point-to-point communications, the author is not aware of there being such a constraint on SOA in general and so this mismatch only applies to web services. The other two mismatches that do not apply, *lit 3* and *lit 4* both refer to the architectural elements in the system being either correct web service components or connectors, in this case they would need to be reworded to relate to SOA instead and the relevant changes made in the type checking rules.

Moving forward, the mismatches identified in this chapter can now be used as part of the specification for the design of an enhanced web service architectural style, which is the subject of the next chapter.

Chapter 5

Enhanced Web Service Architectural Style

Chapter 4 showed that there were more architectural mismatches described in the literature than were described and caught by the minimal architectural style presented in Chapter 3. This confirms a need to build an enhanced version of the architectural style to account for them.

This chapter presents the derivation of the enhanced style. It begins by compiling the complete set of mismatches found, bringing together mismatches from the minimal style and those found in the previous chapter. This results in a list of some 27 mismatches.

The remainder of the chapter is dedicated to description of the derivation of the properties, rules and element types proposed to allow detection of the mismatches. It starts with those mismatches detectable simply by considering any pair of connected ports and then moves on to discuss those that can only be discovered by considering the emergent behaviour of the system as whole. Several of the mismatches tackled by this style were also considered by the minimal style, but while a few of the data structures and analysis rules remain from that earlier work, others have been completely reworked to improve both the data structures themselves and also the focus of the results returned. These changes were made possible by making extensive use of the external analysis features that were made available with ACME Studio 3. Apart from revamping some parts inherited from the minimal style, the external analysis allows for much more powerful analysis techniques to be employed in the style than would be possible under the limitations of the Armani predicate language included in ACME Studio. The most notable example is the generation of CSP models of the system, these are passed to an external model checking tool, FDR, before the results are used to capture emergent mismatches that would not be detectable statically.

5.1 Requirements for the Style

Chapter 4 revealed that there were many more mismatches indicated by the literature than were actually detectable using WSDL. The requirements for the enhanced architectural style will be derived by combining the lists in both Tables 4.2 & 4.3 to form the combined set that will be considered, these are shown in Table 5.1. These mismatches are grouped into three sections based upon their type and the system scope that needs to be considered to determine their existence or not. First are listed the mismatches that can be found simply by comparing any pair of attached ports, these are given IDs matching the form cp<x>. The second set are those mismatches that are found by viewing the system as a whole and in this case performing some model generation and checking. These have IDs of the type cc<x>. Finally there are the type checking mismatches that confirm the system is well defined and uses the correct types, these are labelled ct<x>. Each mismatch is listed with its ID, a descriptive name and the sources from which it is derived.

5.2 Defining the Enhanced Style

The description of this style takes place in four parts. The first section, port to port scope, focusses on those combined mismatches, cp1–cp13, that may be found by comparing any pair of attached ports. This section starts by describing one of the biggest changes between this style and the previous incarnation, the way in which the message exchange patterns are represented. This is followed by a description of the properties and rules associated with detection of this set of mismatches.

The next section, component to environment scope, considers the combined mismatches, cc1–cc6, the ones that can only be found by considering the system as a whole. The premise of the model and the CSP assertions used to detect the commission and omission failures that form the basis of the analysis are described. This is followed by adding in the complications related to allowing multiple conversational threads, multiple connections to a single port and approach to modelling a system containing unknown portions.

The final two parts define the architectural element types included in the style and the rules asserting which of these types may be instantiated.

5.2.1 Port to Port Scope

5.2.1.1 Message Exchange Pattern Description

We can see that a great many of the mismatches listed in Table 5.1 relate to or are affected by the message passing behaviour of the components in the system in terms of the order, quantity and data included in the messages. For example *cc1 - Concurrent calls to a non-queueing and non-reentrant port* relates to the number of messages sent to an individual port, while *cp7 - Mismatching data*

ID	*description*	*sources*
	Port to port scope	
cp1	Mismatching message exchange patterns	lit5 & min1 & min4
cp2	Partially matching message exchange patterns	lit5 & lit21 & min2 & min4
cp3	Incorrect binding time of a service provider	lit7
cp4	Differing data continuity assumptions	lit10
cp5	Mismatching data types in a message	lit11
cp6	Mismatching data structure/syntax	lit13
cp7	Mismatching data semantics in a message	lit12
cp8	Mismatching state maintenance assumptions	lit14
cp9	Mismatching state scope assumptions	lit15
cp10	Mismatching failure mode assumptions	lit16
cp11	Mismatching connector creation/destruction assumptions	lit18
cp12	Connection to a non public web service port	lit19 & min5
cp13	Connected ports must share transport and encoding protocols	min3
	Component to environment scope	
cc1	Concurrent calls to a no queuing and non-reentrant port	lit2
cc2	Mismatching conversations	lit6
cc3	Partially matching conversations	lit6 & lit21
cc4	No component has an active thread of control	lit8
cc5	Concurrent threads in a single thread only component	lit9
cc6	Concurrent threads in a non-reentrant port	lit20
cc7	Mismatching process distribution assumpations	lit17
	Type checking	
ct1	Non web service compliant connector	lit4 & min7
ct2	Non web service compliant component	lit2 & min7
ct3	Ports must be well defined	min5
ct4	Components must have correct port types	min6
ct5	Components must be well defined	ct3[1]
ct6	Connectors must be well defined	ct3 & min8 & lit1
ct7	Roles must be well defined	ct3

[1] ct3 existed for ports but no similar conditions existed for the components, connectors or roles, so these were added.

Table 5.1: The combined set of mismatches that will be considered in the design of the enhanced style.

semantics in a message requires knowledge of the data meaning and types included. The approaches taken to representing the properties upon which these mismatches are founded and the rules that will detect them are presented in the following order:

- Mismatches between connected ports, in terms of the message exchange patterns, data semantics and syntax;
- Mismatches between components in the system in terms of the quantity and order of port invocations; and
- Mismatches of properties that are affected by the conversations, such as multi-threading.

The mismatches labelled *cp1, cp2, cp5, cp6* and *cp7*[1] all focus on the messages exchanged between two connected ports and the semantics and syntax of the data included in those messages. To detect such mismatches we require the following information:

- A representation of the patterns of messages passed between two interating ports;
- The semantics of the data in the messages;
- The types of the data included in those messages.

Only one of these aspects was included in the minimal architectural style presented in Chapter 3, specifically the message exchange pattern. While this did facilitate the detection of mismatching message exchanges, the data structures used necessitated repetition of data and were quite verbose in nature. Another weakness of the structure in the context of this enhanced style is that it did not lend itself to representing the longer term conversations between components that are required. The decision was taken to change the data representing the messages and message exchange patterns completely. Previously the messages and message patterns both existed in the same data structure but now these have been separated out into distinct properties.

The message exchange patterns are now expressed using the formal process algebra CSP[2] and each port in the style holds a CSP description of its message passing behaviour, represented as a single string data item as shown in line 7 of Figure 5.1.

It would be entirley possible to represent many of the properties covered in the architectural models using a single CSP model, in fact it is exactly this single CSP model that the external analysis generated, based upon the simpler properties defined in the style. This would, however, require the creator of the model to be familiar enough with the formalism to construct such a model and the assertion statements that inform if the model meets its specification. It is a goal to allow

[1] *cp1*: Mismatching message exchange patterns, *cp2*: Partially matching message exchange patterns, *cp5*: Mismatching data types in a message, *cp6*: Mismatching data structure/syntax, *cp7*: Mismatching data semantics in a message.

[2] An introduction to all the CSP used in this work can be found in Appendix H

```
Property Type TCSP = string;
Property Type TSafeBoolean = Enum {Yes,No};

Port Type PortTWSCommon = {
   ...
   Property SendsFirstMessage : TSafeBoolean;
   Property MessagePattern : TCSP;
   ...
   rule SendsFirstMessagePopulated = invariant SendsFirstMessage == Yes OR SendsFirstMessage == No;
   rule MessagePatternPopulated = invariant MessagePattern != "";
 }
```

Figure 5.1: The property type and properties to hold the CSP representing the message exchange pattern of a port and also the boolean indicating if that port sends the first message or not.

mismatches to be detected while reducing the work required from the architect as far as possible. It is also a goal that the style could be used by a practitioner architect who may not be versed in formalisms such as, in this case, CSP. To have confidence in the results of analysing a formal model, we must first have confidence in the “correctness” of the model itslef. As already stated, this work does not assume that the user has any specialist knowledge of CSP, so to support this the work includes a set of templates that represent the message exchange patterns available to web services.

As an illustration, the WSDL 2.0 *out-in/in-out* message exchange pattern is presented below. The templates have two purposes :

1. be a verified representation of the message passing behaviour of a specific message exchange pattern, thereby increasing confidence in the models produced; and

2. allow for easy linking to represent subsequent behaviour of the component.

To address the first point requires a specification of the message exchange patterns and these can be found in the W3C WSDL descriptions [W3C06c, W3C06f]. In the case of the *out-in* message exchange pattern text description is as follows :

> 1. A message:
>   * indicated by an Interface Message Reference component whose message label is “Out”and direction is “out”
>   * sent to some node N
> 2. A message:
>   * indicated by an Interface Message Reference component whose message label is “In” and direction is “in”
>   * sent from node N
>
> ⋮ ⋮
>
> Any message after the first in the pattern MAY be replaced with a fault message, which MUST have identical direction.

For completeness, the matching *in-out* pattern is described thus:

1. A message:
   - indicated by an Interface Message Reference component whose message label is "In" and direction is "in"
   - received from some node N
2. A message:
   - indicated by a Interface Message Reference component whose message label is "Out" and direction is "out"
   - sent to node N

⋮ ⋮

Any message after the first in the pattern MAY be replaced with a fault message, which MUST have identical direction.

Chapter 3 presented a graphical interpretation of this and the other patterns starting on page 38 but this is repeated in the diagram in Figure 5.2 for convenience. This pattern has esentially two routes through it, the message is received and a response returned or the message is received and a fault message returned. This can be represented in CSP using the following trivial description:

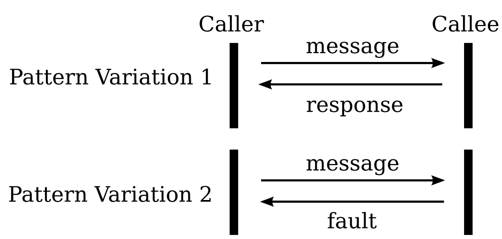


Figure 5.2: WSDL 1.1 Solicit response/Request response and WSDL 2.0 Out-in/In-out message exchange pattern

$$\alpha TRIV\_SPEC\_SOLI = \{Request, Response, Fault\}$$

$$\begin{aligned} TRIV\_SPEC\_SOLI \ \widehat{=}\ & Request \rightarrow (Response \rightarrow Stop \\ & \qquad \Box\ Fault \rightarrow Stop) \end{aligned}$$

This specification, which cannot be proven as it is based upon a natural language description, is arguably a correct representation of the pattern in terms of the messages that may be passed. However it does not represent the direction of those messages, which is important for the analysis described later in this chapter. The message names are expanded to be composed of a *send* part and a *get* part to map properly to the events experienced by each port, so the "request" message is replaced by a "sendReq" and a "getReq" message. This yields the following specification:

$$\begin{aligned}
\alpha SPEC_SOLI &= \{sendReq, getReq, sendRes, getRes, sendFault, getFault\}\\
SPEC_SOLI &\mathrel{\widehat=} sendReq \rightarrow getReq \rightarrow (sendRes \rightarrow getRes \rightarrow Stop\\
&\qquad\qquad\qquad\qquad\qquad \Box\; sendFault \rightarrow getFault \rightarrow Stop)
\end{aligned}$$

The specification describes the messages that would be sent and consumed during a single, correct interaction between a pair of *out-in*/*in-out* ports. From this is is possible to construct the templates for the ports and also a connector process that, when combined, exhibit this exact behaviour. At this point the templates all assume a single interaction and terminate in a *Stop*.

The *out-in* template is as follows:

$$\begin{aligned}
\alpha SOLI &= \{sendReq, getRes, getFault\}\\
SOLI &\mathrel{\widehat=} sendReq \rightarrow SOLI_P1\\
SOLI_P1 &\mathrel{\widehat=} SOLI_P2 \;\Box\; SOLI_P3\\
SOLI_P2 &\mathrel{\widehat=} getRes \rightarrow SOLI_OK\\
SOLI_P3 &\mathrel{\widehat=} getFault \rightarrow SOLI_FAULT\\
SOLI_OK &\mathrel{\widehat=} Stop\\
SOLI_FAULT &\mathrel{\widehat=} Stop
\end{aligned}$$

The *in-out* template is as follows:

$$\begin{aligned}
\alpha REQR &= \{getReq, sendRes, sendFault\}\\
REQR &\mathrel{\widehat=} getReq \rightarrow REQR_P1\\
REQR_P1 &\mathrel{\widehat=} REQR_P2 \;\Box\; REQR_P3\\
REQR_P2 &\mathrel{\widehat=} sendRes \rightarrow REQR_OK\\
REQR_P3 &\mathrel{\widehat=} sendFault \rightarrow REQR_FAULT\\
REQR_OK &\mathrel{\widehat=} Stop\\
REQR_FAULT &\mathrel{\widehat=} Stop
\end{aligned}$$

The templates require a connector process to provide a mapping between the sent message and the recieved message, it also defines the order in which events take place and in effect enforces a send-receive message semantics.

$$\begin{aligned} CONN\_SOLI \mathrel{\widehat{=}} & sendReq \rightarrow getReq \rightarrow CONN\_SOLI \\ & \Box\ sendRes \rightarrow getRes \rightarrow CONN\_SOLI \\ & \Box\ sendFault \rightarrow getFault \rightarrow CONN\_SOLI \end{aligned}$$

Finally then a system comprising of an *out-in* port, an *in-out* port and the connector can be constructed. This composed system is shown to exhibit identical behaviour to the specification by asserting that it refines the specification according to the traces model and also that the specification refines the composed system according to the traces model.

$$PORTS\_SOLI \mathrel{\widehat{=}} SOLI \mathbin{|||} REQR$$
$$COMPOSED\_SOLI \mathrel{\widehat{=}} PORTS\_SOLI \mathbin{|[\, \alpha SOLI, \alpha REQR \,]|} CONN\_SOLI$$

$$COMPOSED\_SOLI \sqsubseteq \mathcal{M}_{UT} SPEC\_SOLI$$

$$SPEC\_SOLI \ \sqsubseteq \mathcal{M}_{UT} COMPOSED\_SOLI$$

After having the message names altered to match those exchanged by the port (discussed later), the message pattern is held in the `MessagePattern` property.

The complete set of message exchange pattern templates and their derivations can be found in Appendix I.

The nature of CSP means that the templates themselves do not explicitly state whether a port expects to send the first message in an exchange or whether it receives it. This problem also existed with the data structure used in the minimal architectural style, so the `SendsFirstMessage` boolean property is retained which is used to orient the message directions in the pattern, line 6 of Figure 5.1.

Finally with respect to defining the message exchange behaviour of the ports, the style needs to check that both the properties described above have been populated with data.

The inclusion of CSP in the style makes use of ACME's interchange language abilities. Unfortunately this does not include any support for the parsing of any parts of the description represented in languages other than ACME. It was considered that writing a CSP syntax checker was outside the scope of the contribution of this work. The result is that the `TCSP` data type is a pseudonym for a string data type and the style checks that the CSP descriptions do not contain the empty string, line 10 of Figure 5.1.

The `SendsFirstMessage` property makes use of the `TSafeBoolean` data type defined in the style that is simply an enumeration of the values `Yes` and `No`. The problem with the native ACME boolean type is that it adopts the value `true` if the property value is not explicitly defined. This makes it impossible to write a rule to confirm that the value of a property has been explicitly defined as the predicate `property == true or property == false`

will always return true, even if `property` has not been assigned a value. However if the same predicate is applied to a property declared as a `TSafeBoolean` as follows

`propertySafe == Yes or propertySafe == No`

then the predicate will return true if and only if `propertySafe` has been assigned the value true, otherwise the rule will show an error resulting from the inability to evaluate the expression. The corresponding rule checking the status of the `SendsFirstMessage` property can be found on line 9 of Figure 5.1. This structure can be found by checking the populated states of all `TSafeBoolean` properties in this style.

5.2.1.2 Message Contents

There are only four pairs of message exchange patterns defined for web services, so if just the `messagePattern` and `sendsFirstMessage` parameters were used to determine if there was a mismatch then there would be a one in eight chance that any two randomly selected ports would match in terms of the number and direction of messages exchanged. Intuitively this is not the case, as the content of the messages also needs to be compatible for the message exchange to be successful. This was acknolwedged in the minimal style where the data included in a message was represented by a syntax token. This token was a string that held a textual description of the message contents. Messages were said to be a match if their syntax tokens were exactly equal. While this scheme allowed for messages to be matched it has two main weaknesses:

- it requires there to be a one-to-one mapping between a message and its description; and
- it hides the message structure from ACME, greatly reducing its ability to type check and parse it.

A further weakness arose from the description of the direction of each message. A direction token was assigned to each message in each port, “inbound” or “outbound”. Due to the simple means applied to determine if the descriptions match, the direction tokens in both ports were required to match also. This meant that the direction tokens in the port that received the first message were always reversed and were therefore considered to be counter intuitive.

Given that mismatches cp5, cp6 and cp7[3] require analysis based upon the semantics and syntactic representation of the data included in the messages passed, a new structure was required to describe them. The goals of the newer structure were both to make the data semantics and representation explicit and available to the analysis functions, and also to to reduce the repetition that was inherent in the earlier structure. The resulting structure, Figure 5.3, is essentially broken down into two parts.

[3] *cp5*: Mismatching data types in a message, *cp6*: Mismatching data structure/syntax, *cp7*: Mismatching data semantics in a message.

```
external analysis EAMessagePatternAndMessageListConcur(thisPort : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.MessagePatternAndMessageListConcur;
external analysis EACentralDataStoreCorrect(thisComponent : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.CentralDataStoreCorrect;
Property Type TStateScopeExpected = Enum {Private,Shared,NoPreference};
Property Type TStateScopeExhibited = Enum {Private,Shared};
Property Type TCentralDataRecord = Record [DatumID : string;
                                           DatumSemantics : TDataSemantics;
                                           DatumScopeExhibited : TStateScopeExhibited; ];
Property Type TDataSemantics = string;
Property Type TDataRep = Enum {SOAP_Int,SOAP_String,SOAP_Float, SOAP_Bool,
                               SOAP_Date, SOAP_Time, SOAP_DateTime};
Property Type TMessage = Record [MessageId : string;
                                 MessageData : Set {TMessageDatum}; ];
Property Type TMessages = set {TMessage};
Property Type TMessageDatum = Record [DatumId : string;
                                      DatumRep : TDataRep;
                                      DatumStateScopeExpected : TStateScopeExpected; ];
Component Type CompTWSCommon = {
   ...
   Property CentralDataRecords : Set {TCentralDataRecord};
   ...
   rule MsgDatumDescribed = invariant EACentralDataStoreCorrect(self);
}
Port Type PortTWSCommon = {
   ...
   Property Messages : TMessages;
   ...
   rule MsgNamesConsistent
       = invariant EAMessagePatternAndMessageListConcur(self);
}
```

Figure 5.3: The properties and types used to represent the messages shared by a component in terms of the data included, their syntax and semantics. Also the definition of the external analysis rules used to check the consistency of the data described in the ports and the component.

The first exists as a property of each port called `Messages`, this stores a description of each message sent or expected to be received by that port. The messages are linked to the messages in the port's `MessagePattern` property by the `MessageId`.

Each message is represented by the type `TMessage` that contains the `MessageId` and a set of `TMessageDatum`. Each element in this set represents a single piece of data included in that message, including a `DatumId` to identify that piece of data, a `DatumRep` describing the syntactic form the data will take and the `DatumStateScopeExpected`, this last item will be described later in this chapter[4].

The second part of the new structure is a property of the component itself named `CentralDataRecords`. This holds a set of `TCentralDataRecord`, a data type containing a `DatumId`, the `DatumSemantics` and the `DatumScopeExhibited`, again more on this last item later. The motivation behind separating out the semantics of a datum from the message description was to allow a single point of declaration for each datum that may then be referenced in many messages if required via the datum ID. A potential bonus of this structure is that it allows the architect to hint at the passage of data items between component ports by using the same ID, though this feature is not utilised in this work.

[4]The type used by `DatumRep`, `TDataRep`, contains six of the SOAP data types only for brevity, there are in fact many more types than this described by the W3C as can be found at `http://www.w3.org/2001/12/soap-encoding`

There are two checks for consistency associated with this structure. The first, evaluated by the rule `MsgNamesConsistent`, focusses on each port declared and confirms that the set of messages named in the `MessagePattern` matches those included in the `Messages` property of that port. It returns true if and only if the sets are identical.

Rule MsgNamesConsistent
$$
\begin{aligned}
&(\forall\, m : \mathit{messageID} \cdot \exists\, M : \mathit{messageID} \\
&\qquad \cdot\ \mathit{inMessages}(m) \\
&\qquad\ \ \wedge\ \mathit{inMessagePattern}(M) \\
&\qquad\ \ \wedge\ \mathit{namesMatch}(m, M)) \\
&\wedge \\
&(\forall\, M : \mathit{messageID} \cdot \exists\, m : \mathit{messageID} \\
&\qquad \cdot\ \mathit{inMessages}(m) \\
&\qquad\ \ \wedge\ \mathit{inMessagePattern}(M) \\
&\qquad\ \ \wedge\ \mathit{namesMatch}(m, M))
\end{aligned}
$$

This analysis makes use of the external analysis feature of Acme Studio as there is no functionality included to allow the extraction of the message names from the CSP message pattern description, which is represented by a single string[5].

The second consistency check is performed between the data declared in the `Messages` set of each port and the `CentralDataRecords`. Here the rule is passed if for every datum declared in each message of every port, there is an entry in the central data records with the same datum ID. This ensures that all data included in the messages can have their semantics examined.

Rule MsgDatumDescribed
$$
\begin{aligned}
&\forall\, m : \mathit{messageID} \cdot \forall\, d : \mathit{datumID} \cdot \exists\, D : \mathit{datumID} \\
&\qquad \cdot\ \mathit{inMessages}(m) \\
&\qquad\ \ \wedge\ \mathit{datumIncludedInMessage}(d, m) \\
&\qquad\ \ \wedge\ \mathit{inCentralData}(D)
\end{aligned}
$$

Again, there is no facility to perform such a check within the Armani predicate language so the rule `MsgDatumDescribed` is evaluated using a plugin developed for the style.

The properties, types and rules described above can be found in the ACME form in Figure 5.3.

With the data structure determined it is now possible to check for mismatches relating to the syntax and semantics of data passed between ports and thus discharge the requirements posed by mismatches cp5, cp6 and cp7[6]. A key decision at this point was what strategy to adopt with respect to matching the data included in the sent message with that in the received message. The three options were, by datum name, by message syntax or by declared semantics. The name is initially attractive but there is as yet no general standard for the naming of the parameters shared. This means it is possible for two components to use different names for the same data item an example of this can be found in the car park scenario used to assess this work in Chapter 6.

The second option was to consider the syntax of the exchanged messages in terms of the order in which data and their types are declared in the service description. This was rejected for two reasons.

[5] The construction of this external analysis, and all others included in this work, can be found in Appendix F. The interested reader is directed there to find the complete Java source code.

[6] *cp5*: Mismatching data types in a message, *cp6*: Mismatching data structure/syntax, *cp7*: Mismatching data semantics in a message.

Firstly SOAP is based upon XML and data included in it are enclosed in XML tags containing the datum name, so data could be extracted from a message by name, meaning that order is not critical. Secondly we consider data representation to be of secondary importance compared to data semantics. For example, if two services agree they they are exchanging a length in metres and one uses a string type "one" and the other uses a float type "1.0" then these services may have their messages mediated and the data representation converted. However if one service is sending a length while the other is expecting a mass, then it does not matter if the data types are matching and the data exchanged are simply not correct.

The chosen approach is to match the datum in the exchanged messages based upon their semantics.

Cp7[7] talks about "mismatching data semantics" as a single problem, however three different cases are identified within this category. Two cases cover scenarios where a sent message is missing one or more items of required data, these are termed "under data". The two cases differ based upon whether there is the possibility that data could be made available or not. The third case is where the sent message contains one or more items of data that are not required by the recipient, this is termed "over data". The rules to determine the existence of all three cases are now presented below.

The first rule checks if there is data expected by the recieving port that is not sent by the sender, but where that data may be available as it, or something with the same semantics, is declared in the central data store. It should only return false if there is data missing and that data may be available, otherwise it should return true.

Rule UnderData1 $\neg(R_s - S_s \neq \phi \wedge (R_s - S_s) \cap S_c \neq \phi)$

$$\begin{aligned} \text{where } R_s &= \text{set semantics expected by the receiving port} \\ S_s &= \text{set semantics actually received} \\ S_c &= \text{set of semantics that could be sent to the receiver} \end{aligned}$$

The second rule checks if there is data expected by the receiving port that is both not sent in the message and not defined in the sender's central data store. In this case the sender simply does not have the required data and cannot therefore send it. This rule should return false if these conditions are met and true otherwise.

Rule UnderData2 $\neg(R_s - S_s \neq \phi \wedge (R_s - S_s) - S_c \neq \phi)$

The third rule looks at the opposite type of mismatch, where data is sent that is not expected by the recipient. This can be determined simply by finding the remainder after subtracting the received message semantics from the sent message semantics. If the remainder is the empty set then there is no extra data so the rule should return true, if the remainder is a non empty set then the rule should return false to indicate a mismatch.

[7] *cp7*: Mismatching data semantics in a message.

Rule OverData $\neg(S_s - R_s \neq \phi)$

The final rule relating to the semantics and syntax of the messages exchanged concerns the data types used to represent each data item in the message. Its purpose is to confirm that where sent and received data have matching semantics, they also have matching data types. An outline of the rule that will check the data type compatibility of the messages would be: for each data item in the sent message, where that data item has a semantic match in the expeced message, their data types must also match.

Rule DataTypesMatch $\forall\, d_s : \mathit{datum} \in \mathit{sentMessage} \cdot \forall\, d_r : \mathit{datum} \in \mathit{receivedMessage}$
$\cdot \mathit{semanticMatch}(d_s, d_r)$
$\Rightarrow \mathit{dataTypesMatch}(d_s, d_r)$

It should be noted that the above four rules are independant of each other and are not mutually exclusive so all combinations of their evaluation to true or false are possible for each message in an interaction. For this reason the rules were separated in the style both in terms of the mismatch they target but also which message in the interaction they examine. For example, there is a rule checking for an over data mismatch in the first message that may be passed between the ports and also other rules checking for the same mismatch in the second, third and fourth messages [8]. The ACME instantiation of these rules can be found in Figure 5.4.

5.2.1.3 Message Mapping

Performing the above analysis requires the descriptions of a pair of messages, one sent by one port and the other received by the other port, such that their properties may be extracted. The mapping between messages sent and messages received is defined by which message exchange pattern each port employs. Table 5.2 contains the data required to map the messages in both ports onto each other. The relations are given in terms of the line number in the CSP template on which the message name will be found and also the direction that message travels, a right arrow indicating 'from the port that sent the first message', a left arrow indicating 'to the port that sent the first message in the exchange'. Only a quarter of the pairings are perfect matches, in all other cases there are one or more messages that are not expected or are not sent, these are indicated by a '-1' on either side of the pairing. In these situations all rules relating to syntax and semantics simply report a 'rule passed' status in ACME Studio as:

- There is only one message so there is nothing to compare; and

[8]The reference here is to the number of messages that are declared for a connected pair of ports not the order in which they may be exchanged. For example, there are four messages defined for an out-optional-in/in-optional-out port pair, *message, response to message, fault triggered by message* and *fault triggered by response*. This is one interpretation of the W3C specification which could also be taken to imply an infinite trace of fault messages triggering fault messages. Interpretations of all the patterns are formally described in Appendix I.

```
external analysis EAMessageDataTypesMatch(firstPort : Element, secondPort : Element, messageNo : int)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.MessageDataTypesMatch;
external analysis EAMessageOverData(firstPort : Element, secondPort : Element,messageNo : int)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.MessageOverData;
external analysis EAMessageUnderData1(firstPort : Element, secondPort : Element,messageNo : int)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.MessageUnderData1;
external analysis EAMessageUnderData2(firstPort : Element, secondPort : Element,messageNo : int)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.MessageUnderData2;

Connector Type ConnTWS = {
   Role role1  = {
   }
   Role role2  = {
   }
   ...
   rule CorrectNumberOfRoles = invariant size(self.ROLES) == 2;
   rule Msg1MessageDataTypesMatch = invariant forall r1 : Role in self.ROLES |
     forall r2 : Role in self.ROLES |
       forall p1 : PortTWSCommon in r1.ATTACHEDPORTS |
         forall p2 : PortTWSCommon in r2.ATTACHEDPORTS |
           (r1 != r2) AND attached(r1, p1) AND attached(r2, p2)
             -> EAMessageDataTypesMatch(p1, p2, 1);
   rule Msg1MessageOverData = invariant forall r1 : Role in self.ROLES |
     forall r2 : Role in self.ROLES |
       forall p1 : PortTWSCommon in r1.ATTACHEDPORTS |
         forall p2 : PortTWSCommon in r2.ATTACHEDPORTS |
           (r1 != r2) AND attached(r1, p1) AND attached(r2, p2)
             -> EAMessageOverData(p1, p2, 1);
   rule Msg1MessageUnderData1 = invariant forall r1 : Role in self.ROLES |
     forall r2 : Role in self.ROLES |
       forall p1 : PortTWSCommon in r1.ATTACHEDPORTS |
         forall p2 : PortTWSCommon in r2.ATTACHEDPORTS |
           (r1 != r2) AND attached(r1, p1) AND attached(r2, p2)
             -> EAMessageUnderData1(p1, p2, 1);
  rule Msg1MessageUnderData2 = invariant forall r1 : Role in self.ROLES |
    forall r2 : Role in self.ROLES |
      forall p1 : PortTWSCommon in r1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in r2.ATTACHEDPORTS |
          (r1 != r2) AND attached(r1, p1) AND attached(r2, p2)
            -> EAMessageUnderData2(p1, p2, 1);
}
```

Figure 5.4: The rules contained in the common connector and port types that are used to check for mismatches in semantics and syntax of the messages shared. For space reasons only the rules targetting the data in the first message in the message exchange pattern are shown, however there are identical rules for the other three messages possible in the current web service patterns.

- This problem will be highlighted when the message exchange patterns themselves are compared, so reporting it here would simply distract from the real problem.

5.2.1.4 Message Exchange Patterns

The next rules presented consider the pattern in which the messages are exchanged between ports, as required to satisfy mismatches cp1 and cp2[9]. Cp1 requires looking for matching messages exchange patterns while for cp2 it is necessary to check for the relaxed condition of partially matching message exchange patterns. Definitions of the conditions under which both of these situations exist were described in Chapter 3 in terms of sets of expected message exchanges. Essentially, patterns were said to be matching if the quantity, direction and contents of the messages described in a pair of ports were identical. A partial match was a relaxation of this, defined as being when the message exchanges expected by one port are a proper subset of another port, where the second port is within our domain of control. However, in the minimal style the message syntax and exchange pattern were recorded in the same data structure and so both were considered when assessing if there was a message exchange match or not.

The data structures have now been separated out allowing the consideration of the number and direction of messages independently of their contents. So analysis of message content mismatches can now be ignored and instead the analysis focusses on the quantity and direction of messages exchanged when considering matching and partially matching message exchanges. This has two effects:

- First it gives a slightly different semantics to partially matching patterns compared to the minimal style. Now they are partially matching if they do not match and one of the ports is within 'our' domain of control; and
- Secondly it gives a more precise indication of the type of problem compared to the minimal style as now the rules can only be failed due to the quantity and direction of messages, not due to the content of the messages.

This leaves the problem of how to assess the quantity and direction of messages each port expects, two options were available at this point. The initial thought was to model check the message exchanges based upon the port CSP, creating a process based upon the connected ports with a connector process that will deadlock whenever a mismatch path is explored. However to create such a connector requires prior knowledge of the mismatching messages, this would mean that the effort required to build and check such a model would be wasted.

[9] *cp1*: Mismatching message exchange patterns, *cp2*: Partially matching message exchange patterns.

			Recieves First Ports			
			ino	*rio*	*reqr*	*ioo*
			(1,req,→)	(1,req,→) (5,flt,←)	(1,req,→) (3,res,←) (4,flt,←)	(1,req,→) (6,res,←) (5,flt,←) (7,flt,→)
Sends First Ports	*noti*	(1,req,→)	(1,1,→)	(1,1,→) (-1,5,←)	(1,1,→) (-1,3,←) (-1,4,←)	(1,1,→) (-1,5,←) (-1,6,←) (-1,8,→)
	roo	(1,req,→) (5,flt,←)	(1,1,→) (5,-1,←)	(1,1,→) (5,5,←)	(1,1,→) (5,4,←) (-1,3,←)	(1,1,→) (5,5,←) (-1,6,←) (-1,8,→)
	soli	(1,req,→) (3,res,←) (4,flt,←)	(1,1,→) (4,-1,←) (3,-1,←)	(1,1,→) (4,5,←) (3,-1,←)	(1,1,→) (3,3,←) (4,4,←)	(1,1,→) (3,6,←) (4,5,←) (-1,8,→)
	ooi	(1,req,→) (4,flt,←) (5,res,←) (9,flt,→)	(1,1,→) (5,-1,←) (4,-1,←) (9,-1,→)	(1,1,→) (4,5,←) (5,-1,←) (9,-1,→)	(1,1,→) (5,3,←) (4,4,←) (9,-1,→)	(1,1,→) (5,6,←) (4,5,←) (9,8,→)

Table 5.2: This table shows the mapping of messages between different message exchange pattern pairings.
Next to each abbrieviated message pattern name are one to four message descriptions, showing the line in the CSP template on which that message can be found, the meaning of the message and the direction it takes. The meanings abbreviations are 'req' = initial request, 'res' = response message, 'flt' = fault message. An arrow pointing to the right indicates the message will be sent by "sends first" port, an arrow pointing to the left indicates it will be sent by the "receives first" port.
The body of the table shows the correct mappings, by line number, of the messages included in each of the CSP templates. For example, "(5,6,←)" indicates the message on the fifth line of the "sends first" port is mapped to the sixth message of the "receives first" port. The left arrow indicates the message originates from the "receives first" port. A -1 indicates there is no message in one pattern associated with the message in the other pattern.
The message exchange pattern name abbreviations are as follows, *ino*: In-Only; *rio*: Robust-In-Only; *reqr*: Request-Response; *ioo*: In-Optional-Out; *noti*: Notification; *roo*: Robust-Out-Only; *soli*: Solicit-Response; *ooi*: Out-Optional-in.

ID	*Msg.*	*Orig.*	*Msg.*	*Orig.*	*Msg.*	*Org.*
T1	req	ob				
T2	req	ob	res	ib		
T3	req	ob	flt	ib		
T4	req	ob	res	ib	flt2	ob

Table 5.3: A table showing the possible traces between an out-optional-in port and an in-optional-out port. The *ID* is simply an identifier, *Msg.* gives a short version of the message name and *Orig.* describes which port sends the message. "ob" = outbound, i.e. the port that sends the first message, "ib" = inbound, i.e the port that receives the first message.

ID	*Msg.*	*Orig.*	*Msg.*	*Orig.*
T1	req	ob		
D1	req	ob	flt	ibd

Table 5.4: A table showing the one matching trace and the one divergent trace when a notification port is paired with a robust-in-only port. Note here the direction of the last message of the divergent trace D1, the label "ibd" tells us that the inbound port desires this message but the outbound port does not send it.

The approach adopted takes advantage of there being only four types of outbound (sends the first message) ports and four types of inbound (receives the first message) ports. This gives a total of 16 sensible combinations of ports, where 'sensible' means a pair containing an outbound port and in inbound one. This small number means that it is possible to predetermine all traces each pair can witness. Such a trace is presented in Table 5.3 where all traces possible for a matching pair of ports consisting of an out-optional-in port along with an in-optional-out are shown, demonstrating that there are four traces this pair of ports would witness. In this case if there were no message content mismatches, these two port message exchange patterns could be described as matching.

A different situation occurs if two ports are connected that are not a natural pair, for example a one-way port with a robust-in-only, the traces of which are shown in Table 5.4. Here we find there is a single trace upon which both ports agree, this is labelled T1, but there is a second divergent trace labelled D1. A divergent trace allows the representation of behaviour expected by one port that is not expected by the other, in this case it is the sending of the fault message by the robust-in-only port which is not expected by the one-way port. It should be noted that while in this instance the divergent trace was only one event longer than the common trace, all messages are recorded. The complete set of these tables of traces can be found in Appendix G.

It should be noted that while in this instance the divergent trace followed the correct trace and added a single event to the end, if there is a sequence of messages occuring after a correct trace then they will all be recorded. In the parlance of Anderson and Lee [AL81] the first message in the expected trace of one port that is not in the others is the point at which the fault is activated. For this reason the traces could stop at the first divergent event but for completeness the entire trace is included.

```
external analysis EAMessageExchangePatternsMatch(thisConnector : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.MessageExchangePatternsMatch;
external analysis EAMessageExchangePatternsPartiallyMatch(thisConnector : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.MessageExchangePatternsPartiallyMatch;

Port Type PortTWSCommon = {
   ...
   Property InOurControlDomain : TSafeBoolean;
   ...
   rule InOurControlDomainPopulated
       = invariant InOurControlDomain == Yes OR InOurControlDomain == No;
}
Connector Type ConnTWS = {
   ...
   rule CorrectNumberOfRoles = invariant size(self.ROLES) == 2;
   rule OnePortSendsFirstMessage = invariant exists r : Role in self.ROLES |
     forall p : PortTWSCommon in r.ATTACHEDPORTS |
       attached(r, p) -> p.SendsFirstMessage == Yes;
   rule OnePortReceivesFirstMessage = invariant exists r : Role in self.ROLES |
     forall p : PortTWSCommon in r.ATTACHEDPORTS |
       attached(r, p) -> p.SendsFirstMessage == No;
   rule MessageExchangePatternsMatch
       = invariant EAMessageExchangePatternsMatch(self);
   rule MessageExchangePatternsPartiallyMatch
       = invariant EAMessageExchangePatternsPartiallyMatch(self);
}
```

Figure 5.5: The rules contained in the common connector that are used to check for mismatches in the message exchange patterns.

Using the complete set of traces it is possible to determine if there are mismatching assumptions about the quantity and direction of messages exchanged by examining the message exchange pattern ID that is included as the first line. This, along with the `inOurControlDomain` safe boolean characteristic that all ports possess, allows us to determine the mismatch status of any two connected ports according to the following statements:

Rule MEPMatch $\neg \mathit{divergentTracesBetween}(\mathit{port}1, \mathit{port}2)$

Rule MEPPartialMatch $\neg \mathit{divergentTracesBetween}(\mathit{port}1, \mathit{port}2)$

$$\begin{aligned} &\vee (\mathit{divergentTracesBetween}(\mathit{port}1, \mathit{port}2) \\ &\quad \wedge (\mathit{inOurControlDomain}(\mathit{port}1) \\ &\qquad \vee \mathit{inOurControlDomain}(\mathit{port}2))) \end{aligned}$$

The two rules should be considered in tandem to determine the type of mismatch, if any, that is discovered. Table 5.5 shows the pass/fail status of each rule and the meaning that should be inferred in terms of the degree of match. All other conditions are considered to be a mismatch. The ACME relating these rules can found in Figure 5.5.

5.2.1.5 State Scope

Mismatch cp9[10] concerns the scope each component associates with each data item, essentially whether it is private to the thread or session that sends it or is visible to any other threads or

[10] cp9: Mismatching state scope assumptions.

	Match	*Partial Match*	*Mismatch*
MEP Match	✓	X	X
MEP Partial Match	✓	✓	X

Table 5.5: Here the results returned from the two rules focussed on the message exchange patterns of each port are related to the degree of match or mismatch that exists between the two ports. A ✓ indicates the rule returned a true result and a 'X' indicates it false result.

sessions in the receiving component. From this, two values are suggested for the characteristic, `Private` and `Shared`.

There is however no clear rule for determining a partial match between such values. For example, if a piece of data is required to be private to a session, such as a user's account details, then this clearly should not be visible to other sessions in that component. At the same time, data that should be shared, such as the availability of a particular parking space in an on-line parking space manager, must be visible to interested threads for correct operation. It may be the case though that a component has no preference about the scope applied to a piece of data it communicates. For example a public weather service may not have such a preference, it is therefore unrealistic to force it to align with one statement or the other. Therefore the style allows a third value to be assigned to data a component communicates and this is `NoPreference`. This value has the semantics implied by its name and will match with either a private or shared value.

The rule checking this characteristic for each datum defined in an interface with the central data store of the connected component is as follows:

Rule StateScopesMatch $\forall\, d_e : datum \in sentMessage$

$$\begin{aligned}
&\cdot\, \exists\, d_c : datum \in oppositeComponentCentralData\\
&\cdot\, semanticMatch(d_s, d_c)\\
&\qquad \Rightarrow d_s.stateScopeExpected == d_c.stateScopeExhibited\\
&\qquad\quad \vee\, d_s.stateScopeExpected == NoPreference
\end{aligned}$$

The ACME portion of this rule along with the additional data types and properties required to support it can be found in Figure 5.6.

5.2.1.6 Data Continuity

Mismatch cp4[11] considers the continuity of data in the system. The literture discusses this characteristic as describing whether a component will have data available either continuously or sporadically, thus this was initially contemplated as being a characteristic of the component. This was rejected however due to the possibility that an architect could describe the entire set of web services an organisation provides in a single WSDL document and also as a single component in this environment.

[11] *cp4*: Differing data continuity assumptions.

```
external analysis EAStateScopesMatch(thisConnector : Element
   ,firstPort : Element,secondPort : Element)
   : boolean = uk.ac.ncl.cjg.ws_enhanced.StateScopesMatch;

Connector Type ConnTWS = {
  ...
  rule StateScopeAssumptionsMatch = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
    forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
      attached(role1, p1) AND attached(role2, p2)
        -> EAStateScopesMatch(self, p1, p2);
  ...
}

Property Type TStateScopeEhxibited = Enum {Private,Shared};
Property Type TStateScopeExpected = Enum {Private,Shared,NoPreference};
```

Figure 5.6: The rule in the connector calling the external analysis to test state scope and the declaration of the property types it uses.

This combined system could provide some services that constantly make fresh data available, such as a wind speed and direction monitor, while others could be sporadic, such as the total sunlight for a given date. Thus the property was moved into the port descriptions. This allows the above set of services to be described.

While a connected pair of ports declaring themselves as `Continuous` or a pair stating `Sporadic` as their property value could be considered to be a match, there appears to be no general rule to determine where different values do not constitute a potential problem. For example, a client port declared as sporadic requesting stock information from a service port declared as continuous should be acceptable, however a sporadic port sending safety related data to a port that expects it continuously would not appear to be satisfactory. Therefore the rule reviewing the model for this mismatch can only be passed where the data continuity values of the connected ports are equal, in all other cases the potential mismatch must be flagged to the user.

Figure 5.7 sets out the two rules required by this mismatch. One rule confirms that the property is populated in each port while the second confirms that an attached pair of ports exhibit the same value.

5.2.1.7 Failure Modes

The failure assumptions and behaviour of the components in a system are the focus of mismatch cp10[12]. Of the literature drawn upon, only DeLine [DeL99] refers to component failure assumptions but does not propose any set of values the characteristic may adopt. There is however the taxonomic work of Avizienis *et al.*[ALRL04] from which a set of five failure mode domains is extracted, {`ContentFailures, EarlyTimingFailures, LateTimingFailures, HaltFailures, ErraticFailures`}.

While other mismatches, and their associated analysis, described in this work aim to discover potential faults such that they may be removed, here it would unrealistic to say that we may discover

[12] *cp10*: Mismatching failure mode assumptions.

```

Port Type PortTWSCommon = {
  ...
  Property DataContinuity : TDataContinuity;
  ...
  rule DataContinuityPopulated =
    invariant DataContinuity == Sporadic
           OR DataContinuity == Continuous;
}

Connector Type ConnTWS = {
  ...
  rule MatchingDataContinuityAssumptions = invariant forall r1 : Role in self.ROLES |
    forall r2 : Role in self.ROLES |
      forall p1 : PortTWSCommon in r1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in r2.ATTACHEDPORTS |
          (r1 != r2) AND attached(r1, p1) AND attached(r2, p2)
            -> p1.DataContinuity == p2.DataContinuity;
}

Property Type TDataContinuity = Enum {Sporadic,Continuous};
```

Figure 5.7: The port property and type describing data continuity and the ACME rule testing for a mismatch.

and remove all failure modes. In this case then the intention is for components to be explicit about their failure modes and also the assumptions they make about the failure modes of other components attached to them. A mismatch is then said to occur when a component may exhibit a failure mode that an attached component does not assume it will. At the same time, it is not considered a mismatch to assume a component may exhibit a type of failure that, due to internal error handling, it will not exhibit.

It is acknowledged that a single web service component in this style may, in fact, be composed of multiple different software components each providing part of its functionality. In respect of this it is assumed that each sub component could both exhibit different failure behaviour and also assume different failure modes of the other components in our system. Therefore the style represents failure behaviour or assumptions not at the component level but rather includes it on a port by port basis.

In the style then a `FailureModesExpected` and a `FailureModesExhibited` property is defined in each port. These properties are both sets that may hold the failure modes listed previously. A mismatch occurs if the following predicate rule, **FailureModeAssumptions**, does not evaluate to true. This rule along with the associated data structures can be found in their ACME form in Figure 5.8.

Rule FailureModeAssumptions $P1.FM_x \subseteq P2.FM_e \wedge P2.FM_x \subseteq P1.FM_e$

where $P1$ and $P2$ = connected ports

FM_x = set of failure modes exhibited by this port

FM_e = set of failure modes assumed by the connected port

This work treats the failure mode names as tokens only, it assumes that the system developers have a shared understanding of their meaning. It also assumes that the analysis of the components

```

Port Type PortTWSCommon = {
  Property FailureModesExpected : TFailureModes;
  Property FailureModesExhibited : TFailureModes;
}

Connector Type ConnTWS = {
  rule FailureModeAssumptions = invariant forall r1 : Role in self.ROLES |
    forall r2 : Role in self.ROLES |
      forall p1 : PortTWSCommon in r1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in r2.ATTACHEDPORTS |
          (r1 != r2) AND attached(r1, p1) AND attached(r2, p2)
            -> (isSubset(p1.FailureModesExhibited, p2.FailureModesExpected))
           AND (isSubset(p2.FailureModesExhibited, p1.FailureModesExpected));
}

Property Type TFailureMode = Enum {ContentFailures, EarlyTimingFailures,
                                   LateTimingFailures, HaltFailures,
                                   ErraticFailures};
Property Type TFailureModes = Set {ws_enhanced_01.TFailureMode};
```

Figure 5.8: The port properties and types describing failure modes and the rule in the connector testing for a mismatch.

leading to the statements of which failure modes it may exhibit and which it can handle when exhibited by other components is performed by some means.

5.2.1.8 Connector Binding Time

Mismatches cp3 and cp11[13] pertain to the attachment of connectors to ports. Specifically when an attachment may be made in terms of the software life-cycle but also which of the parties associated by a connector may have created it or may destroy it.

The former mismatch, as discussed in Chapter 4 concerns the binding time of a service provider. The style model allows for three types of component, *client*, *intermediary* and *service*. The primary difference between these types lies in the interfaces their ports belong to. The ports on a client component must all be part of the client interface, the ports on a service component must all contribute to its service interface while an intermediary should have ports representing both interfaces. Thus the rules relating to this mismatch are located within the port definitions so they target the correct interfaces.

The purpose of the rule is to check the point in the software lifecycle that the port expects to bind to another. The model enumerates four points in the cycle as follows:

write-time: when the process is designed and written;

compile-time: when the process is compiled and linked;

instantiation-time: when an instance of the process is constructed; and

run-time: when the instance is running, this may include idle time waiting for communications.

[13] *cp3*: Incorrect binding time of a service provider, *cp11*: Mismatching connector creation/destruction assumptions.

```
Port Type PortTWSCommon = {
  Property BindTime : TBindTime;
  ...
}

Port Type PortTWSClient extends PortTWSCommon with {
  rule BindingTimePopulated =
    invariant BindTime == Write
           OR BindTime == Compile
           OR BindTime == Instantiation
           OR BindTime == Run;
}

Port Type PortTWSService extends PortTWSCommon with {
  rule StatedBindingTime =
    invariant BindTime == Instantiation
           OR BindTime == Run;
}

Property Type TBindTime = Enum {Write,Compile,Instantiation,Run};
```

Figure 5.9: The properties and type describing when a port will bind and the rules checking their values are suitable for the port type.

The fault to detect is when a service port is pre-bound to a specific set of clients or addresses. In this case while it may be discovered in a search for services it would not be possible to utilise the service as it would not allow itself to bind to the new client. It follows then that the port should be as late-binding as possible, so in this model a service port that reports binding at run-time would meet this criteria.

Also, if a new instance of a service is created in response to a client request then this would exhibit the required late binding. To acknowledge this the style also allows services to bind at instantiation time.

The rule, **StatedBindingTime**, is located within the style description of the `PortTWSService` and can be described using the following predicate. At the same time the ACME fragment showing the actual rule and associated data structures can be found in Figure 5.9.

Rule StatedBindingTimeMismatch $BindTime = Run \vee BindTime == Instantiation$

Mismatch cp11[14] forces consideration of which of the ports attached by a connector may either create or destroy that connector. This is approached in terms of a statement of rights to either create or destroy the connector rather than commitment to or prohibition to do either at any particular point in time[15].

In the model, four properties are declared in each port that will make explicit these statements of rights, these properties are `BindingSelfAdd`, `BindingSelfRemove`, `BindingOtherAdd` and `BindingOtherRemove`. In the naming of these properties, `self` refers to the port in which the property is stated and `Other` means a port attached to the other end of the connector. At the same time, `Add` implies the ability to create the connector (binding) between the two ports while `Remove`

[14] *cp11*: Mismatching connector creation/destruction assumptions.

[15] The issue of when connector changes can be made is discussed in Chapter 7 on Future Work.

indicates the potential to destroy the connector (unbinding).

Creating a connector means a binding between the ports or a willingness to exchange messages, this is opposed to the act of sending or receiving messages which may or may not happen as a consequence of the branches in the conversation tree taken. Destroying a connector, conversely, is where the ports will no longer expect message traffic to pass between them.

A key point to understanding this property is that while it is stated on a port by port basis, it is not necessarily the individual ports that create or destroy the willingness to participate in an exchange of messages. Rather it will be the component itself or possibly another port. However as a component may embody a number of different functionalities and it is possible that each may have different binding and unbinding characteristics, it is required that each port state its own properties. Here the ports effectively say 'with respect to this port, the component expects to be able to create/destroy connections'.

For example, a weather application may choose to connect to a free weather service that provides two ports. Port S handles subscription to/unsubscribing from the service, while port M sends out the regular weather updates. Here, after exchanging messages with port S the client will be willing to accept weather updates from port M. Thus in the model the connector between the client and M would now be created even though no messages have yet passed. Also the client may unsubscribe from the service at any time, effectively destroying the connector between itself and M as it is no longer willing to receive messages. In this case both the client ports have the right to create connections and they also have the right to destroy the connections[16]. At the same time, we can also imagine a situation where an application assumes it has the right to create a connector on a port and will do so in the normal course of events but that it also allows another component to do so if required. In this case we must allow for the possibility that a component may not actually have a preference about whether the other component believes it has the right to connect or not as either are acceptable to it.

To support these options a two value logic is applied for describing a components own rights to create and destroy a connector, specifically the terms `May` and `MayNot` are used. A pseudo three value logic *****ref here***** is used to describe assumptions about the other components rights, specifically the values `May`, `MayNot` and `Either`. The predicate rule, **ConnectorCreationDestruction**, for detecting a mismatch accounts for both the situation where a component makes a specific assumption about the other component's rights and the situation where it does not. In the first case the rule confirms that the values assumed by each component are equal, while in the second case it allows a component to apply the `Either` to the other so long as it applies the value `May` to itself. This final assertion ensures that at least one of the components will have the right to create/destroy the

[16]In this slightly simplified example the service is not allowed to cancel a subscription, though this could also be captured by giving the service the right to destroy connections.

connector.

Rule ConnectorCreationDestruction

$$\begin{aligned}
&\big(P1.\mathit{BindingOtherAdd} == P2.\mathit{BindingSelfAdd}\\
&\quad \vee (P1.\mathit{BindingOtherAdd} == \mathit{Either} \wedge P1.\mathit{BindingSelfAdd} == \mathit{May})\big)\\
&\wedge\big(P1.\mathit{BindingOtherRemove} == P2.\mathit{BindingSelfRemove}\\
&\quad \vee (P1.\mathit{BindingOtherRemove} == \mathit{Either} \wedge P1.\mathit{BindingSelfRemove} == \mathit{May})\big)\\
&\wedge\big(P2.\mathit{BindingOtherAdd} == P1.\mathit{BindingSelfAdd}\\
&\quad \vee (P2.\mathit{BindingOtherAdd} == \mathit{Either} \wedge P2.\mathit{BindingSelfAdd} == \mathit{May})\big)\\
&\wedge\big(P2.\mathit{BindingOtherRemove} == P1.\mathit{BindingSelfRemove}\\
&\quad \vee (P2.\mathit{BindingOtherRemove} == \mathit{Either} \wedge P2.\mathit{BindingSelfRemove} == \mathit{May})\big)
\end{aligned}$$

The above rule requires a companion to guard against the entry of nonsensical data such as all four properties being assigned the value `No`, the `TSafeBoolean` equivalent of false. Thus a sanity check that each port expects that at least one of them can create the connector and at least one of them can destroy it is added[17]. The second predicate **SaneConnectorCreationDestruction** capturing this sanity check is recounted below. The ACME versions of both these rules, along with the supporting data structures and rules confirming that properties are populated can be found in Figure 5.10.

Rule SaneConnectorCreationDestruction

$$\begin{aligned}
&(P1.\mathit{BindingSelfAdd} == \mathit{May}\\
&\quad \vee P2.\mathit{BindingSelfAdd} == \mathit{May})\\
&\wedge(P1.\mathit{BindingSelfRemove} == \mathit{May}\\
&\quad \vee P2.\mathit{BindingSelfRemove} == \mathit{May})
\end{aligned}$$

5.2.1.9 End Points

The final analysis performed by comparing pairs of connected ports looks at the mismatches labelled cp13 and ct3[18]. These relate to mismatching end point protocols and missing service port descriptions respectively. Both of these were included in the minimal architectural style, they are briefly repeated here as no changes were deemed necessary.

The endpoint protocols are defined by a pair consisting of a network transport protocol and a message encoding, each pair prescribing the protocols supported by a particular endpoint. As with the minimal style the rules determine if a pair of connected ports have at least one common endpoint

[17] The situation where a connector is created and then exists for perpetuity is considered to be highly unlikely and so is guarded against in the style.

[18] *cp13*: Connected ports must share transport and encoding protocols, *ct3*: Ports must be well defined.

```
    Property Type TConnCreationDestructionAssumption = Enum {May, MayNot, Either};

Port Type PortTWSCommon = {
  ...
  Property BindingSelfAdd : TConnCreationDestructionAssumption;
  Property BindingSelfRemove : TConnCreationDestructionAssumption;
  Property BindingOtherAdd : TConnCreationDestructionAssumption;
  Property BindingOtherRemove : TConnCreationDestructionAssumption;
  ...
  rule BindingSelfAddPopulated = invariant
    BindingSelfAdd == May
      OR BindingSelfAdd == MayNot;
  rule BindingSelfRemovePopulated = invariant
    BindingSelfRemove == May
      OR BindingSelfRemove == MayNot;
  rule BindingOtherAddPopulated = invariant
    BindingOtherAdd == May
      OR BindingOtherAdd == MayNot
        OR BindingOtherAdd == Either;
  rule BindingOtherRemovePopulated = invariant
    BindingOtherRemove == May
      OR BindingOtherRemove == MayNot
        OR BindingOtherRemove == Either;
}

Connector Type ConnTWS = {
  ...

  rule ConnectorCreationDestruction = invariant
    forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
      forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1) AND attached(role2, p2)
          ->  (p1.BindingOtherAdd == p2.BindingSelfAdd
                OR(p1.BindingOtherAdd == Either AND p1.BindingSelfAdd == May))
          AND (p1.BindingOtherRemove == p2.BindingSelfRemove
                OR(p1.BindingOtherRemove == Either AND p1.BindingSelfRemove == May))
          AND (p2.BindingOtherAdd == p1.BindingSelfAdd
                OR(p2.BindingOtherAdd == Either AND p2.BindingSelfAdd == May))
          AND (p2.BindingOtherRemove == p1.BindingSelfRemove
                OR(p2.BindingOtherRemove == Either AND p2.BindingSelfRemove == May));

  rule SaneConnectorCreationDestruction = invariant
    forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
      forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1) AND attached(role2, p2)
          ->  (p1.BindingSelfAdd == May OR p2.BindingSelfAdd == May)
          AND (p1.BindingSelfRemove == May OR p2.BindingSelfRemove == May);
}
```

Figure 5.10: Properties describing if a port assumes it can create or destroy a connector and whether it assumes the other port can create or destroy the connector.

```
Port Type PortTWSCommon = {
  Property EndPointList : TEndPoints;
  ...
  rule EndpointListPopulated = invariant size(EndPointList) > 0;
  ...
}

Connector Type ConnTWS = {
  rule EndpointProtocols = invariant forall r1 : Role in self.ROLES |
    forall r2 : Role in self.ROLES |
      forall p1 : PortTWSCommon in r1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in r2.ATTACHEDPORTS |
          (r1 != r2 AND attached(r1, p1) AND attached(r2, p2))
          -> size(intersection(p1.EndPointList, p2.EndPointList)) > 0;
}

Property Type TEndPoint = Record [Transport : TLegalTransportProtocols;
                                  Encoding : TLegalSoapVersions; ];
Property Type TEndPoints = Set {TEndPoint};
Property Type TLegalSoapVersions = Enum {SOAP1_1,SOAP1_2};
Property Type TLegalTransportProtocols = Enum {HTTP1_0,HTTP1_1};
```

Figure 5.11: The properties and types describing the protocols supported by a web service endpoint, also the rule to confirm that a connected pair of ports share a common protocol pair.

protocol, and use that to conclude that there is not a mismatch. There is no built in mechanism to allow ACME Studio to inform which protocol pair(s) the connected ports have in common, so the determination of which to employ would be left to the user.

The predicate rule **EndpointProtocols** capturing this is shown below, with the ACME version to be found in Figure 5.11.

Rule EndpointProtocols $P1.\mathit{EndPointList} \cap P2.\mathit{EndPointList} \neq \phi$

The second mismatch brought forward from the minimal style, checking that a service type port is well defined, requires confirmation that a number of descriptive properties are populated correctly. The first of these data items is the `EndPointAddressList`. This is a set holding the network addresses at which the endpoints of a port may be found. The model requires both that the list be populated, otherwise the port may not be accessed, and also that there is an address for each endpoint defined for that port. The predicate rules **EndPointAddressPopulated** and **EachEndpointProtocolAddressed** address these requirements.

Rule EndPointAddressPopulated $\mathit{size}(\mathit{EndPointAddressList}) > 0$

Rule EachEndpointProtocolAddressed $\mathit{size}(\mathit{EndPointAddressList}) = \mathit{size}(\mathit{EndPointList})$

The final data checked is to confirm that the service port is published for discovery, this addresses mismatch cp12[19] by asserting that all service ports must be referenced in at least one WSDL document.. To confirm this a check that the port has an entry in the `WSDLDocRefs` property is performed.

[19] cp12: Connection to a non public web service port.

```
Port Type PortTWSService extends PortTWSCommon with {
  ...
  Property WsdlDocRefs : TWsdlDocs;
  Property EndPointAddressList : TEndPointAddresses;
  ...
  rule EndPointAddressPopulated =
    invariant size(EndPointAddressList) > 0;
  rule EachEndpointProtocolAddressed =
    invariant size(EndPointAddressList) == size(EndPointList);
  rule HasWSDL = invariant size(WsdlDocRefs) > 0;
}

Property Type TWsdlDocs = Set {string};
Property Type TEndPointAddresses = Set {string};
```

Figure 5.12: The properties and types describing how a service port is made discoverable along with rules to confirm the properties are populated.

This property is a set as the same port may be referenced in multiple descriptions. The predicate rule **HasWSDL** is shown below, with the ACME version of this and the previous two rules to be found in Figure 5.12.

Rule HasWSDL $size(WsdlDocRefs) > 0$

5.2.2 Component to Environment Scope

The remaining mismatches identified in Chapter 4 can only be detected by considering the system as sa whole and not by focussing on individual pairs of connected ports. This is because these mismatches all either affect or are a result of the emergent behaviour of the system in terms of the messages shared and the logical threads of control they witness. Some of the rules consider properties of the whole system, such as determining if the system starts live and will do something. Others take the focus of an individual component interacting with its environment, such as checking for mismatching conversational expectations.

Mismatches cc2 and cc3[20], relate to mismatching and partially matching conversations. Conversations in this context refer to all the messages exchanged between two or more components while conducting their business. This means if we considered a simple service that has only a single port, then the conversation would be identical to the message exchange pattern of that port. It could also consist of many more messages if the service requires a client to log in on one port before browsing a directory on another and making purchases on yet another. In either case a mismatch exists if the components disagree on either the quantity, direction of content of the messages exchanged during these conversations. Rules to check for mismatches relating to both the syntax and semantics of the messages exchanged between two ports have already been described. To avoid a “double jeopardy” situation these results are ignored when checking the conversations, in effect the following rules

[20] *cc2*: Mismatching conversations, *cc3*: Partially matching conversations.

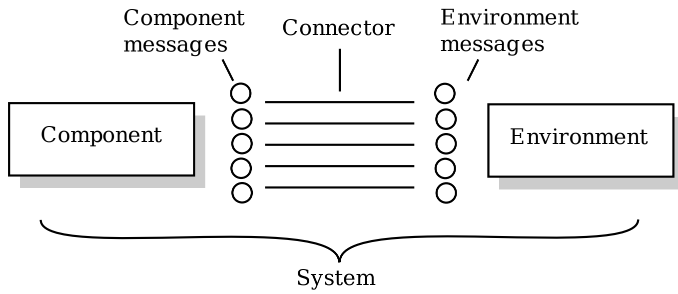


Figure 5.13: For the conversational analysis the rules take the view point of each component in turn and consider its interactions with its directly connected environment in terms of the messages it may send and receive. The connector both translates the message names between components and enforces a send-receive semantics that CSP does provide.

say "if all the messages exchanged were semantically and syntactically correct, then these are the conversation mismatches that would exist".

While the conversation(s) that take place are a property of the system as a whole, the rules presented all take the focus of each component in the system in turn. That is, they consider the problems that occur at the interface between the "current" component and its environment. Here the environment consists of only those components that directly exchange messages with the component in focus. This gives a view of the system as depicted in Figure 5.13, where we have the messages sent and received by the component and those sent and received by the environment. These two sets of messages acknowledge that correctly matched components may use different names to represent the same message, this requirement to translate the message names and the means by which it is achieved is vital to the analysis as shall be seen later.

The SHARD [Pum99] guidewords *commission* and *omission* provide us with the faults we need to look for relating to message exchanges[21]. Commission describes an extra or unexpected message, while omission is the term for a missing message. In terms of the analysis view described above, commission relates to an extra message sent by either the focussed component or its environment while omission is a message that either the component or the environment will not receive as it was never sent. The analysis view allows us to consideronly commission and omission events occurring with respect to the component in focus as the rules are eventually applied to each and every component in the system and in this way we will also visit each component in the environment as well.

A key point of this approach is that it explores all branches of the conversation tree a system can visit and it is done without making any assumption about the internal decision making process of any component. The term cooperative choice is used to describe this approach which means that if

[21]SHARD also includes the guidewords *early* and *late* but the literature search did not highlight actual timings for message exchanges as being potential architectural mismatches. This only leaves message sequences to be considered, in which case a late message could also be described as an omitted message followed by a commission of another message and, mutatis mutandis, the same is true for an early message.

a component has to make a choice about which port it will listen on for the next incoming message then it will defer that choice until the message arrives. This is done as it allows the analysis to visit all points in the conversation tree and know that if an expected message does not arrive or if an unexpected message does then it is due to the underlying choreography in the system and is not caused by an internal choice or an internal decision such as a time-out.

5.2.2.1 Basic CSP System Model

Before the analysis is described, the basics of the CSP model of the system that it will generate must be introduced.

The model consists of a number of component processes and the vital connector process. The components themselves are composed from the CSP descriptions of the ports that were described earlier in this chapter and also a central CSP description of the component. The central CSP essentially describes how many threads of control exist in a component, which ports on that component are willing to interact initially and can also be used to support the chaining together of ports to describe acceptable conversation trees. It starts with a single process that has the same name as the component and then defines a number of processes representing the number of threads of control it will contain. This central CSP is described in Appendix I.3 with an illustrative examples given.

Essentially the combination of the central CSP along with the CSP descriptions in each port describe the basic structure of the conversational trees expected by a component. It is required that there are no duplicated process or event names in the composed system, so each name is prepended with the name of the component and port. The components are then composed into an interleaved process that forms the bulk of the model. Below we see a system composed of three components called *COMP1*, *COMP2* and *COMP3*.

$$SYS \mathrel{\widehat{=}} COMP1 \mathbin{|||} COMP2 \mathbin{|||} COMP3$$

While there are many connectors in the architectural model of the system, there is only a single connector process in the CSP model. The connector simply consists of a number of simple processes that have two purposes. Firstly they provide a name translation service to associate the sent message with the received message, secondly they allow the forcing of a send-receive semantics by separating the sent message from the received message and giving them an order. This latter property is the part vital to the analysis as is described shortly.

$$
\begin{aligned}
CONN \mathrel{\widehat{=}}\ & msg1Send \rightarrow msg1Receive \\
\Box\ & msg2Send \rightarrow msg2Receive \\
\Box\ & msg3Send \rightarrow msg3Receive \\
& \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
\Box\ & msgNSend \rightarrow msgNReceive
\end{aligned}
$$

The final step is to create parallel processes consisting of the components and the connector, with the two synchronised on the messages sent and received. This ensures that a component may not send a message unless the connector is waiting for one to be sent and that the connector cannot deliver a message unless the target component and port are ready for that to occur.

5.2.2.2 Basic Conversational Analysis: Commission

The basic conversational analysis has two parts to it to capture two issues from the viewpoint of the component in focus, these are:

- component sends an unexpected message (commission)
- component does not receive an expected message (omission)

These two conditions are symmetrical to the environment receiving an unexpected message or not sending an expected message. So as each component in the system will at some point be the component in focus and will at some other points form part of the environment interface, the analysis will cover all commission and omission related issues in this way.

The first part of the analysis targets the commission events. This is conducted by performing a deadlock analysis on the composed system model. A deadlock occurs when the CSP model reaches a point where it is unable to perform any further events so the trace of the model cannot proceed. The style uses the single connector, previously described, to force a deadlock when a message is sent but the target port is not ready to receive it. This property is guaranteed as once a message is placed upon the connector the only event the connector can perform is to deliver that message. The connector is synchronised on all send and receive events with the relevant ports, so if the port is not ready to receive that specific message then the connector cannot proceed. Then as no further messages may be placed on the connector the whole system deadlocks.

Detection of a deadlock highlights a problem but it does not immediatly point to the component in focus as being the sender. To determine this we need to examine each deadlock trace, as there may be many, returned by the model checker. As no message passing events can occur once the system is deadlocked we know that the last message in the trace caused the deadlock and so if that message is one that the component in focus sends then the failure occurs because that component is sending an unexpected message. If the message causing the deadlock was not sent by the component in focus, then another component is sending the unexpected message and the analysis will report the mismatch when that component becomes the one in focus.

Once it is determined that this component causes the commission failure the style is then obliged to ascertain how to report this to the architect. Recall that both mismatching conversations and partially matching conversations were listed in the mismatch table (Table 5.1, mismatches cc2 & cc3). This is because the commission occurance could be avoided by either not attempting to send the message or allowing the target component to receive it. These could be implemented by altering the conversation of one or both components involved, a task only possible if one or both of the components has the value true assigned to the `ComponentInOurControlDomain` property. The style acknowledges this by including two rules examining for commission events, one detects when a commision occurs but neither component can be altered, this is reported as a mismatch. The other detects when a commission occurs and one or both of the components can be altered, this is reported as a potential partial match[22].

As both of these rules are implemented using external analysis it is possible to return extra detail regarding the traces that lead to the deadlock in the form of a text file. Details of the text file output and a description of the data included for both this and every other external analysis that makes use of the feature can be found in Appendix F.

Two rules are included in the style for detecting these commission events. The **CommissionMismatch** rule informs the architect if there is a commission event and neither component is desclared as being in our control domain. **CommissionPartial** fails if there is a commission failure and either of the components involved is under our control. The ACME declarations of these rules, external analysis and associated properties can be found in Figure 5.14.

Rule CommissionMismatch $\exists\, dt : deadlockTrace$

$$\cdot \neg inOurControlDomain(senderLastMessage(dt)) \wedge \neg inOurControlDomain(receiverLastMessage(dt))$$

Rule CommissionPartial $\exists\, dt : deadlockTrace$

$$\cdot\, inOurControlDomain(senderLastMessage(dt)) \vee inOurControlDomain(receiverLastMessage(dt))$$

5.2.2.3 Basic Conversational Analysis: Omission

Deadlock can only tell us about events that actually occur in a trace, it cannot tell us about events that do not occur, this means a different method is required to detect the omission events representing

[22] The rule reports a potential partial match as we cannot determine autonomously from this model if the required changes can be made when considering the purpose and business rules of each component.

```
external analysis EACommissionMismatch(thisComponent : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.CommissionMismatch;
external analysis EACommissionPartialMatch(thisComponent : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.CommissionPartialMatch;
external analysis EAChoiceGroupsHaveChoiceMaker(thisComponent : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.ChoiceGroupsHaveChoiceMaker;

Component Type CompTWSCommon = {
  Property CentralProcessDescription : TCSP;
  Property ComponentInOurControlDomain : TSafeBoolean;

  ...

  rule CentralProcessDescribed = invariant CentralProcessDescription != "";
  rule ComponentInOurControlDomainDescribed
    = invariant ComponentInOurControlDomain == Yes
             OR ComponentInOurControlDomain == No;
   rule CommissionMismatch = invariant EACommissionMismatch(self);
   rule CommissionPartialMatch = invariant EACommissionPartialMatch(self);
   rule ChoiceGroupsHaveChoiceMakers = invariant EAChoiceGroupsHaveChoiceMaker(self);
}
```

Figure 5.14: The rules implementing the commission analysis required.

unfulfilled expected messages. The traces refinement concept of the CSP formalism is employed for this purpose. A model A is a traces refinement of model B if all the traces of A are also traces of B[23]. Essentially this is used to confirm that a model does nothing that is not allowed by its specification.

The following work is based upon two assertions:

- the CSP model of a component describes the behaviour that component expects to witness in terms of messages sent and recieved; and
- the CSP model generated of the whole system of components and connectors describes all conversations, in terms of messages that can actually occur in that system.

The analysis is based upon an assertion in the CSP model that the system model, after hiding all events and messages other than those the component in focus sends and receives is refined by the CSP model of the component itself.

$$\alpha \mathrel{\widehat{=}} \mathit{messagesNotInComponentInterface}$$
$$\mathit{SYS} \setminus \alpha \sqsubseteq \mathcal{M}_{UT}\,\mathit{COMPONENT}$$

This assertion can only be true if the component can experience all branches of its expected conversations, in which case the traces of the system model, hiding all other messages, will be identical to the traces of the component. If the system does not allow any branch of the conversation to be explored then the system model will not contain a trace including that branch so the refinement will fail as the component model will contain that trace.

This analysis cannot be performed in isolation as it may return potentially false negative results if the system deadlocks and prevents one or more branches of the component's conversation tree

[23]A more detailed description of traces refinement and its semantics can be found in Schneider [Sch00].

being followed. This is termed a potential false negative as until the deadlock is resolved it is not possible to tell if the refinement failure is a real problem or not. To differentiate between potential omission failure and real ones the deadlock trace results found previously are utilised. A refinement failure is deemed to be a potential false negative if the trace leading to the omitted message can be found in its entirity, in order, with no other messages from that component's interface, in one or more of the deadlock traces. This means that the component is able to follow a conversational branch up to the point where is is ready to receive that message, but there is some fault in the system that is preventing it from happening. The argument here is that if the deadlock did not occur then the system may proceed to a point where the omitted message is sent and that particular refinement failure no longer exists.

Once the deadlock is removed, one of three situations may occur:

- A later and preivously unreached deadlock may appear and the potential false negative will still exist;
- the refinement failure will still occur and with no relevant deadlocks it will then be counted as a genuine omission failure and will be flagged to the architect for rectification;or
- the refinement failure will not occur, indicating the message would be sent and received and that it was originally a false negative.

Again, the style allows for the possibility that the component expecting the message may be declared to be in our control domain as described earlier. If it is, then the omission event is reported as a potential partial match, if not then it is reported as a mismatch[24].

At the same time the refinement assertion will also highlight messages the component expects to send but cannot due to some earlier deadlock. Unlike the omission failures, refinement failures involving sent messages can only be caused by the connector being deadlocked at the point where the port was ready to send the message, as otherwise there would be at least one interleaving during model checking that would allow the message to be placed and so the refinement check would not be failed. A true commission failure will show up in the deadlock analysis previously described, so refinement failures ending with an outgoing message from that component are ignored.

Two rules are included in the style for detecting omission events. **OmissionMismatch** reports a missing expected message where the waiting component is not under our control while **Omission-Partial** reports a missing message where the waiting component is under our control. The ACME declarations of these rules, external analysis and associated properties can be found in Figure 5.15.

[24] It is possible to determine both the sending and receiving component in a commission event so both are included when considering a partial match. However if multiple connectors are attached to an inbound port then it is not currently able to determine which component(s) may have been expected to send a message in an omission event, so only the recieving component is considered for the purposes of partial match.

```
external analysis EAOmissionMismatch(thisComponent : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.OmissionMismatch;
external analysis EAOmissionPartialMatch(thisComponent : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.OmissionPartialMatch;
external analysis EAChoiceGroupsHaveChoiceMaker(thisComponent : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.ChoiceGroupsHaveChoiceMaker;

Component Type CompTWSCommon = {
  ...

   rule OmissionMismatch = invariant EAOmissionMismatch(self);
   rule OmissionPartialMatch = invariant EAOmissionPartialMatch(self);
}
```

Figure 5.15: The rules implementing the omission analysis required.

Rule OmissionMismatch $\exists\, rt : \mathit{refinementFailureTrace}$

$$\begin{aligned} &\cdot \neg \exists\, dt : \mathit{deadlockTrace} \\ &\cdot \mathit{traceContains}(\mathit{cropLastMessage}(rt), dt) \\ &\quad \Rightarrow \neg \mathit{inOurControlDomain}(\mathit{receiverLastMessage}(rt)) \end{aligned}$$

Rule OmissionPartial $\exists\, rt : \mathit{refinementFailureTrace}$

$$\begin{aligned} &\cdot \neg \exists\, dt : \mathit{deadlockTrace} \\ &\cdot \rightarrow \mathit{traceContains}(\mathit{cropLastMessage}(rt), dt) \\ &\quad \Rightarrow \mathit{inOurControlDomain}(\mathit{receiverLastMessage}(rt)) \end{aligned}$$

5.2.2.4 Cooperative Connectors

The inclusion of the intermediary component type acknowledges that some services may be dependant upon others. If such a service is provided by a different administrative domain then this opens up the possibility that the architect may not know the components and toplogy of the architecture on the far side of that component. This could result in a situation where the model includes ports that are not attached to any connectors or other components. This is problematic in two ways. Firstly, it will not be clear to an observer whether an unattached port represents the gateway to unknown portions of the system or simply an incomplete model. Secondly the analysis described above relies heavily upon the connector deadlocking to trap failures and stop further processing, if a port is not attached to the common connector then it will not be bound to halt when the connector locks, weakening the analysis.

The compromise here is to add a second type of connector to the style, called `ConnTWSCooperative`. This connector has only a single role in its description, no properties or rules and is termed 'cooperative' as it represents a perfectly matched component on its other, virtual, end. While it includes no analysis rules in itself it does contribute to the CSP models of the sytem by adding further events to the connector. For example, if a cooperative connector is attached to a port which sends the message called '`request`' and then expects either '`response`' or '`fault`' to be returned then the

```
Connector Type ConnTWSCooperative = {
    Role role1  = {
    }
}
```

Figure 5.16: The entire description of the ConnTWSCooperative connector type showing that it only contains a single role and no rules or properties. Its purpose is to inform both the user and the external analysis that this is the end of our knowledge of the system and, for the purposes of analysis, we should assume everything beyond it works perfectly.

golden connector would add the following branches to the common connector `CONN`.

$$
\begin{aligned}
& CONN = ... \\
& \qquad \square\ request \rightarrow CONN \\
& \qquad \square\ response \rightarrow CONN \\
& \qquad \square\ fault \rightarrow CONN
\end{aligned}
$$

Assuming the connector is not already deadlocked or in the middle of delivering another message then this addition is always willing to perform any of the message send or receive events of the port without the risk of deadlocking the connector as there are no events following any of them. However if the connector is already deadlocked then the port will not be able to send or receive any further messages which could comprimise the previously described analysis. The ACME description of this connector type, showing that its only feature which is a single role with no properties or rules, can be seen in Figure 5.16.

5.2.2.5 Stubborn Connectors

Previously in this chapter the common connector type used in this style has been described. This connector type is used to represent the conduit between every pair of known interacting ports in the system. Then in the previous sub-section the cooperative connector was introduced to acknowledge that there may be interactions taking place with elements that are outside of the scope of the system being modelled. Both of these types are based on the assumption that a connection exists for every port in the system, even if the element on the other end is not known.

There may be occasions however when there are ports in the system to which no connection is made, an example of this will be shown later in Section 6.3.1 when describing one of the scenarios used to assess this work. In the scenario there exists a port to which there is no obvious partner, in terms of the data exchanged, to connect it to, attaching a normal connector between the ports highlights this mismatch. Another approach would be to attach a cooperative connector to the port, however this is inappropriate also as it would represent a connection to a component outside the visible system where that component is well matched in terms of data exchanged and the message exchange pattern, neither of which is true in the scenario.

To acknowledge this, a third type of connector is included in the style, called `ConnTWSStubborn` to reflect the effect it has on the behaviour of the attached port. Recall that one of the concerns with using a cooperative connector was that it represents an unseen port that reacts as a port expects, let's call this example port 'A'. In the resulting CSP model of the system this means that if port A were to send a message, then the unseen port would be willing to receive it. At the same time the unseen port is willing to send any messages that port A expects to receive. The result of this is that if the behaviour of the system is such that port A should interact with another, then it will interact as it expects. However, given that no such connection exists in the scenario any messages it sends will have no destination and there is no "other" port to send the messages it expects. In response to this, the behaviour of the stubborn connector is to block any interactions assumed by the port, the goal being to highlight the system traces that lead to it needing to interact. The blocking behaviour takes two forms, both involving the connector process in the CSP model of the system.

- The first form considers messages the port expects to send. Here the connector allows the message to be placed onto the network and then performs a CSP *Stop* event. This will have the effect of deadlocking the system immediately after the message is placed and will cause the send attempt to be detected as a commission event.

- The second form considers a message the port expects to receive. In this case we allow the connector to deliver the message, but only after it witnesses a *faux* event. Since this event cannot happen, as the style assumes the architect will not use it as a message name, this means the port will never receive the message. A trace leading to this state would be revealed by an omission event.

An example of the CSP connector described above is shown below. In this example the port expects to send the *request* message and receive both the *response* and *fault* messages.

$$
\begin{aligned}
&CONN = ... \\
&\qquad \Box\, request \rightarrow Stop \\
&\qquad \Box\, faux \rightarrow response \rightarrow CONN \\
&\qquad \Box\, faux \rightarrow fault \rightarrow CONN
\end{aligned}
$$

Thus by using the stubborn connector type the architect may explicitly highlight a port for which there are no connections and be able to see the effect it has on the system behaviour.

It should be noted that while the facility exists to instantiate the stubborn connector type, the external analysis defaults to adding this type of connector to any port in the system that has no connectors attached to it. In this way a system that contains no connections will not be able to

```
Connector Type ConnTWSStubborn = {
    Role role1  = {
    }
}
```

Figure 5.17: The ACME description of the stubborn port type. This type has only a single role and no rules as there will not be a pair of ports attached to compare. Its purpose is to explicitly highlight a port that is not connected to any other and to allow the CSP models to report traces in which such a port would expect to interact.

witness any successful message exchanges, instead it will fail both the commission and omission rules. This is arguably the correct result of analysing such a system.

The ACME description of this connector type can be found in Figure 5.17. The type has only a single role since it should only connect to a single port. Also it does not perform any of the analysis of the common connector as there is not a pair of ports to compare.

5.2.2.6 Multiple Connections

Unlike the minimal style where the majority of the rules existed in the connector and only considered the point to point scope defined by its instances, the enhanced style contains rules that are affected by the topology of the system in terms of its message passing behaviour. Specifically these rules are the ones associated with the conversations a component will have with the components it is connected to. This means that the models assessed by those rules need to represent the possible interplay of two or more connections to a single port.

The style allows both client ports and service ports to attach to multiple connectors. E-mail is a good example of where both of these situations may occur, a client application may connect to separate servers hosting the user's different accounts, while a mail server may service multiple clients simultaneously.

An example of such a topology is shown in Figure 5.18, where a simple e-mail client is connected to two e-mail servers. The three ports on it could represent login, download mail and logout operations. If it is assumed that the e-mail client can only interact with one mail server at a time then it follows that if it performs a successful operation on mail server A then it should perform the download mail and log out operations on that server before it attempts to perform any operations on any other server. To put this another way, it is not expected that a successful login operation on mail server A followed by an attempt to download mail from server B would be successful.

While a combination of the central component CSP and port CSP can easily represent the constraints on the order in which the client invokes the login/download mail/logout ports, it gives no indication regarding their dependancies in terms of which server(s) they can be directed to. To support this the style includes another level in the hierarchy of port types. There are similar specialisms for both the client and service port types, one specialism for ports that attach to a single

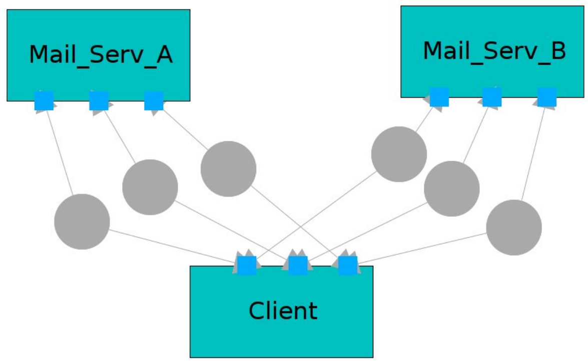


Figure 5.18: Simple e-mail client attached to two e-mail servers and showing how multiple connectors may be attached to a single port.

connector only, and another that supports one or more connector attachments.

`PortTWSClientSingle` and `PortTWSServiceSingle` extend their respective parent types by initialising values of `BindingCardinalityMin` and `BindingCardinalityMax` both to 1. This has the effect of constraining that port type to only allow a single connector to be attached without failing the rule `CardinalityOfAttachmentsOK` defined in the `PortTWSCommon` type. As there is only a single attachment allowed on this type of port no further modifications are necessary.

The other type of specialist ports are the `PortTWSClientUnicast` and `PortTWSServiceUnicast`. Both of these port types are endowed with two new properties that are required to describe the depancies of choice between ports. The first property is a string type in each port called `ChoiceGroup`. Ports that share a name are considered to be in a group where they all have connections to the same set of components and if one chooses to send a message to a specific component then the following ports will also send their messages to that component. The actual value of this choice group property has no significance beyond defining a unique group in a component. In Figure 5.18 all three ports on the client component would have the same value as they share the same choice of component.

The second property added is a safe boolean called `ChoiceGroupMaker`. As the name implies this property determines whether a particular port is one that is allowed to make a choice about which component it and other members of the group will communicate with (`ChoiceGroupMaker = Yes`) or whether it is a port that must follow the choice of another (`ChoiceGroupMaker = No`). In the e-mail example, the login port is defined as the choice maker, while the other two ports have to follow its choices.

Both of these properties are included to allow the effects of communicating with each of the possible components to be examined and so the characteristics represented by these properties need to be included in the CSP models that are produced. To do this the external analysis makes three changes to the model when compared to a system where only single connectors are attached to each port. The first change is to the CSP describing the message exchange pattern of the port. By default all the pattern templates assume that only a single connector will be attached to the port, so at each point in the process where a message is sent or received. However now the model needs to allow for that message to be sent to or received from any of the connected components. This is achieved using the external choice operator and by also copying the message, renaming it to include the name of the component it was sent to or received from. These new message names are then included in the connector and mapped to the approriate attached componet to ensure it is delivered to or received from the correct one. The log in port from the e-mail client, which uses the solicit-response CSP template, would intially look like this:

$$\begin{aligned}
&LOG_IN && \widehat{=}\ logIn \rightarrow LOG_IN_P1 \\
&LOG_IN_P1 && \widehat{=}\ LOG_I N_P2 \,\Box\, LOG_IN_P3 \\
&LOG_IN_P2 && \widehat{=}\ logInResult \rightarrow LOG_IN_OK \\
&LOG_IN_P3 && \widehat{=}\ fault \rightarrow LOG_IN_FAULT \\
&LOG_IN_OK && \widehat{=}\ DOWNLOAD \\
&LOG_IN_FAULT && \widehat{=}\ LOG_IN
\end{aligned}$$

After manipulation we see that the original messages have been replaced with choices of a message to or from each connected component :

$$\begin{aligned}
&LOG_IN && \widehat{=}\ (logInMail_Serv_A \rightarrow LOG_IN_P1 \\
& && \quad \Box\, logInMail_Serv_B \rightarrow LOG_IN_P1) \\
&LOG_IN_P1 && \widehat{=}\ LOG_IN_P2 \,\Box\, LOG_IN_P3 \\
&LOG_IN_P2 && \widehat{=}\ (logInResultMail_Serv_A \rightarrow LOG_IN_OK \\
& && \quad \Box\, logInResultMail_Serv_B \rightarrow LOG_IN_OK) \\
&LOG_IN_P3 && \widehat{=}\ (faultMail_Serv_A \rightarrow LOG_IN_FAULT \\
& && \quad \Box\, faultMail_Serv_B \rightarrow LOG_IN_FAULT) \\
&LOG_IN_OK && \widehat{=}\ DOWNLOAD \\
&LOG_IN_FAULT && \widehat{=}\ LOG_IN
\end{aligned}$$

However there is still a need to record the choice made in the above port and to ensure that the two following ports follow that choice. The following process was added to the model to perform this action:

$$\begin{aligned}
CHOICE\ \widehat{=}\ & logInMail_Serv_A \rightarrow CHOICEMail_Serv_A \\
\Box\ & logInMail_Serv_B \rightarrow CHOICEMail_Serv_B
\end{aligned}$$

$$
\begin{aligned}
CHOICEMail\_Serv\_A \mathrel{\widehat=} & \; logInMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; logInMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; logInResultMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; faultMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; downloadMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; downloadResultMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; downloadFaultMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; logOutMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; logOutResultMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; logOutFaultMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A
\end{aligned}
$$

$$
\begin{aligned}
CHOICEMail\_Serv\_B \mathrel{\widehat=} & \; logInMail\_Serv\_A \rightarrow CHOICEMail\_Serv\_A \\
& \Box \; logInMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; logInResultMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; faultMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; downloadMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; downloadResultMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; downloadFaultMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; logOutMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; logOutResultMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B \\
& \Box \; logOutFaultMail\_Serv\_B \rightarrow CHOICEMail\_Serv\_B
\end{aligned}
$$

The choice process "*CHOICE*" in the above example represents the initial quiescent state before any decision has been made as to which component to interact with. Essentially the choice process is initially willing to allow a message to be sent from the login port to either mail server A or B. Once a message has been sent to one of these the process moves to either $CHOICE\_A$ or $CHOICE\_B$ depending on the target of the message and effectively records the choice made by the login port.

The subprocesses have a structure that both allows future choices of target to be made by the choice maker port while also constraining the dependent ports so they only communicate with the current choice of component. The ability to make future choice is provided by replicating the structure of the *CHOICE* process in each of the sub processes. This allows the process to choose to send the logInMail_Serv_A or logInMail_Serv_B message each and every time the conversation thread reaches the login port. The remainder of each subprocess defines the set of messages that the conversation thread is allowed to exchange given that the decision of which component to interact with has been made. Observe that in the case of the $CHOICE\_A$ process the messages sent and

```
Port Type PortTWSClientSingle extends PortTWSClient with {
   rule CardinalityOfAttachmentsOK = invariant size(self.ATTACHEDROLES) == 1;
}

Port Type PortTWSClientUnicast extends PortTWSClient with {
    Property ChoiceGroup : string;
    Property GroupChoiceMaker : TSafeBoolean;
    rule ChoiceGroupPopulated = invariant ChoiceGroup != "";
    rule CardinalityOfAttachmentsOK = invariant size(self.ATTACHEDROLES) > 0;
}

Port Type PortTWSServiceSingle extends PortTWSService with {
   rule CardinalityOfAttachmentsOK = invariant size(self.ATTACHEDROLES) == 1;
}

Port Type PortTWSServiceUnicast extends PortTWSService with {
    Property ChoiceGroup : string;
    Property GroupChoiceMaker : TSafeBoolean;
    rule ChoiceGroupPopulated = invariant ChoiceGroup != "";
    rule CardinalityOfAttachmentsOK = invariant size(self.ATTACHEDROLES) > 0;
}
```

Figure 5.19: The properties and rules pertaining to the cardinality of their bindings. The 'single' type ports must have a single connector attached to be correct while the 'unicast' types must have one or more connectors attached.

received by the download and logout ports are all appended with the name of their target component "Mail_Serv_A", matching the renamed messages in those port's CSP descriptions. The same is true for *CHOICE_B*, just with the target component being "Mail_Serv_B"[25].

The definition of the choice process is performed automatically by the external analysis and each conversation thread defined in the central component CSP is placed in parallel with it, synchronising on each message sent or received by each port in the choice group. In this way each conversation thread in the component can make independent non interfering choices about which component to communicate with.

From an analysis point of view there are no changes between a system with only single connectors attached to each port and those with multiple connections included. The analysis previously described still applies as each message is named such that its intended target or source is identified. This maintains the unique naming of messages that is required for the model to function and also identifies the pair of components between which a commission or omission failure is found.

Having defined ports that make different expectations about the number of attachments they should encounter, the style also includes rules to inform the architect should these expectations be breached. For the 'single' type ports there should be a single connector attached, while the 'unicast' type ports function with one of more ports attached. It should be noted that the style assumes that no ports are left unconnected in a system, so both the rules for single and unicast port types preclude zero attachments. These rules, called `CardinalityOfAttachmentsOK`, can be seen for all four port types in Figure 5.19.

[25]The port CSP descriptions are not shown for the download and logout ports, but they have essentially the same structure as the login port and the same modifications to allow them to send and receive messages from either component A or B.

5.2.2.7 Multi-threading

In the previous section it was shown that the enhanced style supports multiple connectors attached to a single port, also the style includes templates, described in Appendix I.3, that allow multiple conversation threads to be defined in a component. Both of these open up the possibility that a port could experience concurrent attempts to invoke it. This forces the style to support the detection of mismatches cc1 & cc6[26], that consider the number of concurrent threads in a non-reentrant port.

If a port is reentrant then the assumption is that it is able to process concurrent invocations without any undesirable side effects. However if the port is not reentrant then the assumption is that it does not support concurrent invocations and a system is therefore defined as containing a mismatch if such a port is subjected to multiple invocations. The mismatch exists in a specific port if the following rule evaluates to false:

Rule PortReentered $Port.Reentrant == Yes \vee MaxThreadsInPort < 2$

The first clause in the predicate is trivial to assess as there is a property, `Reentrant`, in the common port description. This safe boolean type property is given the value `Yes` if the port supports concurrency, otherwise is should be given the value `No`.

The second clause requires the use of external analysis as once again it can only be determined by model checking the system. For the purposes of the analysis, the rule does not need to return a value giving the exact maximum number of threads simultaneously in the port, instead it simply just needs to return a boolean value relating to the second clause above. A true value implies there was never more than one thread in the port at any time while false indicates that two or more threads in the port occurred at some point during the model checking.

The basis of this anlysis is that the points in the message exchange pattern templates indicating the entrance and exit of a conversational thread from that port are identified. Using the request response message exchange pattern as an example, the receipt of the request message would indicate the entrance of the conversational thread and the sending of either the response or fault messages would indicate it leaving. To allow detection of the event where more than one thread exists in the component a thread monitor process is introduced that synchronises with the thread entry and exit points of the message exchange pattern.

$$THREAD\_MON\_O \mathrel{\widehat{=}} request \rightarrow THREAD\_MON\_1$$

$$\begin{aligned} THREAD\_MON\_1 \mathrel{\widehat{=}} & \; request \rightarrow THREAD\_MON\_2 \\ & \Box \; response \rightarrow THREAD\_MON\_O \\ & \Box \; fault \rightarrow THREAD\_MON\_O \end{aligned}$$

$$THREAD\_MON\_2 \mathrel{\widehat{=}} multiThreads \rightarrow Stop$$

[26] *cc1*: Concurrent calls to a no queuing and non-reentrant port, *cc6*: Concurrent threads in a non-reentrant port.

The *THREAD_MON* process has three states. It starts as *THREAD_MON_0* as no ports can start by containing a thread. The first instance of *request* message moves the process to *THREAD_MON_1* indicating that the port now contains a thread. While in this state, if either a *response* or *fault* message is witnessed then the thread monitor moves back to the zero thread state, however if a *request* message is seen then the process moves to *THREAD_MON_2*. This last state indicates that there are two threads concurrently in that port, this is indicated by generating a *multiThreads* event and then stopping. The *Stop* event is used once multiple threads have been detected as we know that the situation can occur and there is no need for further model checking and by stopping this process, which is synchronised with the port, we aim to expedite the termination of the model checking process.

To actually detect that concurrent events occured the style once again uses the refinement feature of CSP. A specification process is generated that contains all the messages sent and received by the port

$$\begin{aligned} \mathit{THREAD\_SPEC} \mathrel{\widehat{=}} & \ \mathit{request} \rightarrow \mathit{THREAD\_SPEC} \\ & \square \ \mathit{response} \rightarrow \mathit{THREAD\_SPEC} \\ & \square \ \mathit{fault} \rightarrow \mathit{THREAD\_SPEC} \end{aligned}$$

Finally, the analysis checks for the condition occurring using the following assertion that the specification is refined by the system containing the port and the thread monitor after hiding all messages not involved with that port.

$$\mathit{THREAD\_SPEC} \sqsubseteq \mathcal{M}_{UT} \mathit{SYSTEM} \setminus \mathit{allOtherMessages}$$

If there are one or more traces where concurrency could occur in the port being examined then the above assertion will fail, this result can be returned by the external analysis to be fed into the port reentrance rule. The rules and properties supporting the detection of a reentrance mismatch can be found in Figure 5.20.

5.2.2.8 Complications and Interleaving

The above example was presented using the request-response message exchange pattern, in which there are clear points where the conversational thread could be said to enter and exit the pattern. This, however, is not the case for all the message exchange patterns web services may employ.

While in the request-response pattern it could be assumed that the conversation enters the port when it receives the first message and leaves it when it sends the response, this does not apply to its counter part, the solicit-response pattern. In solicit-response the first message it witnesses is the one

```
external analysis EAConcurrentCallsToThisPort(thisPort : Element)
    : boolean = uk.ac.ncl.cjg.ws_enhanced.ConcurrentCallsToThisPort;

Port Type PortTWSCommon = {
  Property Reentrant : TSafeBoolean;
  ...
  rule PortReentered =
    invariant Reentrant == Yes
           OR EAConcurrentCallsToThisPort(self) == true;
  rule ReentrantPopulated = invariant Reentrant == Yes OR Reentrant == No;
  ...
}
```

Figure 5.20: The properties used to describe if a port is reentrant and supports concurrency and the rules calling the external analysis to determine if it occurs or not.

it sends out at the begining of the exchange, from this it could be inferred that the conversational thread was present in that port before the message was sent as it will have contributed to the construction of that message. To represent this faithfully in the CSP model would require an event in the template before the sending of the first message, however this is not possible as it would break the conversational analysis by interfering with the cooperative choice it requires[27].

The solution lies in the fact that conversational threads in our components are interleaved and explore all combinations of traces. If we imagine a simplified solicit-response pattern where no fault message is allowed, then adding an event to represent the entry of the thread, *incThread*, before the first message would yield the following process:

$$PORT \mathrel{\widehat{=}} incThread \rightarrow request \rightarrow response \rightarrow Stop$$

If we then contruct a system with two instances of *PORT* interleaved and extract the set of complete traces we end up with:

$$\begin{array}{ll}
Trace\ 1 & incThread \rightarrow request \rightarrow response \rightarrow incThread \rightarrow request \rightarrow response \\
Trace\ 2 & incThread \rightarrow request \rightarrow incThread \rightarrow response \rightarrow request \rightarrow response \\
Trace\ 3 & incThread \rightarrow request \rightarrow incThread \rightarrow request \rightarrow response \rightarrow response \\
Trace\ 4 & incThread \rightarrow incThread \rightarrow request \rightarrow response \rightarrow request \rightarrow response \\
Trace\ 5 & incThread \rightarrow incThread \rightarrow request \rightarrow request \rightarrow response \rightarrow response
\end{array}$$

If the events *incThread* and *response* are used to indicate the points at which the conversation enters and leaves the port then we see that all traces except Trace 1 contain a section where both processes contain a thread simultaneously and so the port would have experienced concurrency.

Abstracting away from the above model and assuming that the thread enters the solicit-response port when it sends the first messages, the now redundant *incThread* event can be removed resulting

[27] There would now be an event after the start of the template and before the first message, this means the first message is effectively hidden from any choice of which port the conversation should follow and so that decision cannot be made cooperatively.

in the process *PORT2*.

$$PORT2 \widehat{=} request \rightarrow response \rightarrow Stop$$

Interleaving two instances of *PORT2* will yield the following traces:

$$\begin{array}{ll} Trace\ 1 & request \rightarrow response \rightarrow request \rightarrow response \\ Trace\ 2 & request \rightarrow request \rightarrow response \rightarrow response \end{array}$$

If the *request* event is now used to indicate entry of the thread and *response* to indicate its exit, then Trace 1 shows an interleaving where no concurrency issues exist, while Trace 2 shows an execution where concurrency will occur.

This shows us that, as the analysis is simply looking for the existence of a trace in which concurrency occurs, either model of the port could be used and would return the same result. So the style adopts the following abstraction, any port type that sends the first message will be modelled as receiving the conversational thread at the point where that message is sent. A similar argument can be used to ignore any housekeeping that takes place after the last message in a pattern and simply use that last message, if it exists, to indicate the exit of a thread.

5.2.2.9 No Explicit Pattern Termination

Two of the message exchange patterns, notification and in-only contain only a single message in their structure. Applying the abstraction discussed above it is possible to determine the point at which the thread enters the port but there is no event to indicate the exit of the thread. It is vital that there be separate events to increment and decrement the thread count so the model has a finite, non zero length of time for the thread to be in the port. In this case the only option was to add an aritificial *decThread* event to the CSP template after the message. This achieves the finite period of thread occupancy while not adding or blocking any decisions the conversation can make regarding the path followed. This allows detection of concurrent invocations of the port while leaving the results in terms of system traces unaltered when the new event is hidden.

5.2.2.10 Patterns with Optional Non-explicit Endings

The two message exchange pattern pairs added with WSDL 2.0, robust-out-only/robust-in-only and out-optional-in/in-optional-out pose the same problem, they both contain paths including optional, additional messages. The result of this is that the exchange of the initial message in the pattern does not guarantee the exchange of any further messages. Using the simpler of the two patterns as an example, this means that after the first message has been exchanged, the receiving port may or

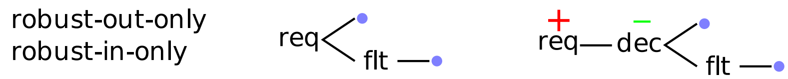


Figure 5.21: The robust-out-only and robust-in-only patterns, on the left before adding the decThread event and with the new event added on the right.

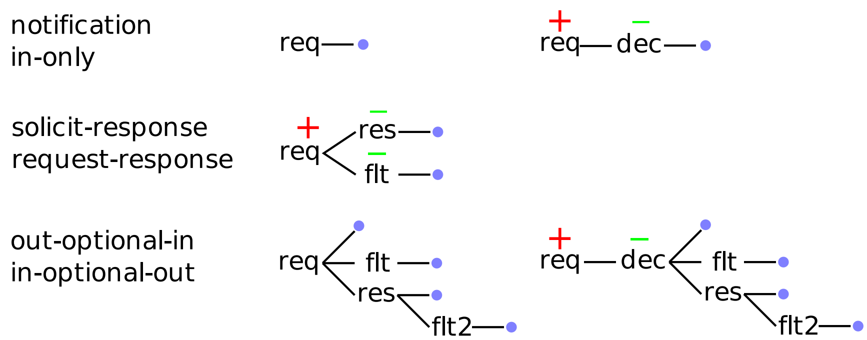


Figure 5.22: The remaining three message exchange patterns shown as trees. On the left the original patterns and on the right those with the additional artificial decThread event. req, res, flt and flt2 are all messages appropriate to each pattern, dec represents the decThread event. The blue dot represents the termination points of the patterns and the red + and green - represent the entry point and exit point of the thread in the CSP models.

may not respond with a fault message. From a thread point of view this means there is no explicit point at which the thread leaves the port.

An initial approach taken was to model a timeout by adding the *decThread* event in any branch that terminated without an explicit message exchange. However this approach introduced deadlocks into the system where one port times out while the other sends the fault message. While it could be argued that this echoes reality as a message received after a timeout could constitute a commission event, it is not in keeping with the approach taken towards analysis which is to cooperatively explore the possible conversations and only report commissions and omissions that are a result of choreography and not performance/timing.

The result of this was that the *decThread* event was moved to be after the first message but before any decision points in the pattern, as shown in Figure 5.21.

This position means that the modelled concurrency critical section is much shorter than the patterns themselves. This is justified by appealing to the argument made earlier regarding the entry and exit of threads before and after the first and last messages. The interleaving model means we get the same result using this shorter critical section as if the critical section were modelled as being the entire length of the pattern, so long as the anlysis is simply interested in the existence of a concurrency event rather than the exact length of that event or the number of traces including it.

Diagrams showing the original patterns along with the new modifed patterns including the artificial *dexThread* events can be found in Figure 5.22.

5.2.3 Architecture Elements

5.2.3.1 Components

Most of the references in this chapter have been to a common web service type that is not intended to be directly instantiated in the model. As with the minimal style there is the desire to distinguish between the three roles a component may adopt and constrain the port types they contain accordingly. The three types of component intended for use are `CompTWSClient`, `CompTWSService` and `CompTWSIntermediary`. The client type is intended to represent the client component that connects to and uses services provided by other components, it is therefore only permitted to host ports that satisfy the type `PortTWSClient`. The service component type, as the name suggests, provides services that other components may discover and use. It is constrained to allow only ports satisfying the type `PortTWSService` to be associated with it. The final type of component allowed is the intermediary, this type can host both client and service type ports. It can act as a go between for other components, perhaps to increase the dependability of service provided as in the Web Service Mediator described by Chen [Che08].

Shown in Figure 5.23 are the component type descriptions. All types extend the common type but include their own rules to tailor the port types they each allow, also ensuring that each component has at least one port. Also shown is the declaration of the client and service port types with an enumerated property to allow the ACME rules to positively distinguish between them. These rules address the mismatch type ct4[28]. The complete hierarchy of component types in this style can be seen in Figure 5.24.

5.2.3.2 Ports

The properties and rules included in the port types have all been included in the previously presented ACME fragments so there is nothing to add here other than to clarify the hierarchy of types. Figure 5.25 shows the hieararchy. Only the `PortTWSClientSingle`, `PortTWSClientUnicast`, `PortTWSServiceSingle` and `PortTWSServiceUnicast` types are intended to be instantiated in a system, their supertypes do not contain all the rules or properties required for proper analysis.

5.2.3.3 Connectors

Finally in the elements is the simple hierarchy of connector types in this style, Figure 5.26. The `ConnTWSCommon` should be used for all connections between ports in the system, the `ConnTWSCooperative` connector type serves to represent unknown portions of the system while the `ConnTWSStubborn` connector makes explicit connections that we know will not exist.

[28] *ct4*: Components must have correct port types.

```
Port Type PortTWSClient extends PortTWSCommon with {
  Property InInterface : TInterfaces = Client;
  ....
}

Port Type PortTWSService extends PortTWSCommon with {
  Property InInterface : TInterfaces = Service;
  ...
}

Component Type CompTWSClient extends CompTWSCommon with {
  ...
  rule AllClientPorts = invariant forall p : Port in self.PORTS |
    satisfiesType(p, PortTWSClientSingle)
    OR satisfiesType(p, PortTWSClientUnicast);
  rule ComponentHasValidPorts = invariant size(self.PORTS) > 0;
}

Component Type CompTWSIntermediary extends CompTWSCommon with {
  ...
  rule ComponenthasValidPorts = invariant forall p : Port in self.PORTS |
    satisfiesType(p, PortTWSClientSingle)
    OR satisfiesType(p, PortTWSClientUnicast)
    OR satisfiesType(p, PortTWSServiceSingle)
    OR satisfiesType(p, PortTWSServiceUnicast);
  rule ComponentHasClientInterface = invariant exists p : Port in self.PORTS |
    satisfiesType(p, PortTWSClientSingle)
    OR satisfiesType(p, PortTWSClientUnicast);
  rule ComponentHasServiceInterface = invariant exists p : Port in self.PORTS |
    satisfiesType(p, PortTWSServiceSingle)
    OR satisfiesType(p, PortTWSServiceUnicast);
}

Component Type CompTWSService extends CompTWSCommon with {
  ...
  rule AllServicePorts = invariant forall p : Port in self.PORTS |
    satisfiesType(p, PortTWSServiceSingle)
    OR satisfiesType(p, PortTWSServiceUnicast);
  rule ComponentHasValidPorts = invariant size(self.PORTS) > 0;
}

Property Type TInterfaces = Enum {Client,Service};
```

Figure 5.23: The definition of the final component and port types used along with the rules regarding the port types each component type may host.

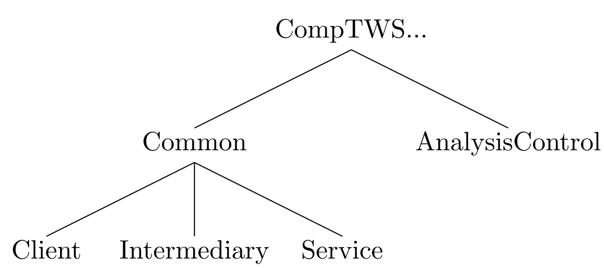


Figure 5.24: The hierarchy of component types in the enhanced style.

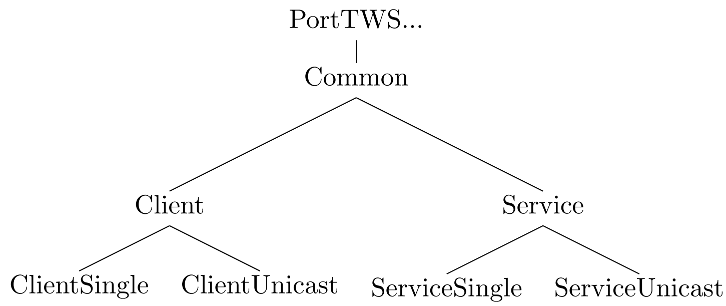


Figure 5.25: The hierarchy of port types in the enhanced style.

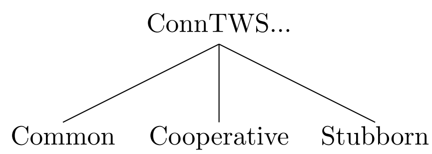


Figure 5.26: The hierarchy of connector types in the enhanced style.

5.2.4 Type Checking

The final rules included in the style serve two purposes. Firstly they assert that all component, port and connectors instantiated in a system must be those defined in this style. Secondly they only allow a subset of all types defined in the style to be instantiated without indicating a fault. The former aspect of the rule disallows the standard ACME component, port and connector types to be instantiated as these by default have no properties and contain no rules to perform the analysis required. The latter aspect acknowledges the hierarchic approach taken in building the style. This means that only the leaf elements in each tree branch contain all properties and rules required by the style and so only these are allowed to exist in the system. These rules directly address mismatch types ct1 & ct2[29] and in doing so they enforce the checking of the remainder of the type related mismatches ct2-ct7[30]

The two rules, one constraining the connector types and the other the component types can be seen in Figure 5.27. The observant reader may note that there is a fourth type of component `CompTWSAnalysisControl` allowed in the nature of components rule. This type is not intended to represent an element in an actual system but is used to allow some control over the external analysis that takes place. As this type is not part of the web service style per se it is not detailed here but is described along with the external analysis it controls in Appendix F.

[29] *ct1*: Non web service compliant connector, *ct2*: Non web service compliant component.
[30] *ct3*: Ports must be well defined, *ct4*: Components must have correct port types, *ct5*: Components must be well defined, *ct6*: Connectors must be well defined, *ct7*: Roles must be well defined.

```
...
rule NatureOfComponents = invariant forall comp : Component in self.COMPONENTS |
  satisfiesType(comp, CompTWSClient)
  OR satisfiesType(comp, CompTWSService)
  OR satisfiesType(comp, CompTWSIntermediary);
  OR satisfiesType(comp, CompTWSAnalysisControl);
rule NatureOfConnectors = invariant forall conn : Connector in self.CONNECTORS |
  satisfiesType(conn, ConnTWS)
  OR satisfiesType(conn, ConnTWSCooperative)
  OR satisfiesType(conn, ConnTWSStubborn);
```

Figure 5.27: Rules asserting the only types of connectors and components that may exist in the system.

5.3 Summary

This chapter started with the compilation of a set of mismatches applicable to web service compositions. These mismatches were then used as guidance for the construction of our enhanced web service architectural style. The style definition was divided up into three separate parts, each targeting a different scope of problem, port to port mismatches, component to environment mismatches and conformity to the style.

Table 5.6 repeats the list of mismatches intended for inclusion in the style and shows in which section they are addressed. The observant reader may have noticed that a small number of the mismatches from the combined set presented in Table 5.6 were not addressed in this style. Specifically these were

cp8 Mismatching state maintenance assumptions;

cc4 No component has an active thread of control;

cc5 Concurrent threads in single thread only component; and

cc7 Mismatching process distribution assumptions.

These items will be discussed under future work in Chapter 7.

Moving on, with the style and its supporting external analysis in place, the work now is to test and evaluate the style and its associated analysis as a tool for detecting the mismatches.

ID	*description*	*Section*
	Port to port scope	
cp1	Mismatching message exchange patterns	5.2.1.4
cp2	Partially matching message exchange patterns	5.2.1.4
cp3	Incorrect binding time of a service provider	5.2.1.8
cp4	Differing data continuity assumptions	5.2.1.6
cp5	Mismatching data types in a message	5.2.1.2
cp6	Mismatching data structure/syntax	5.2.1.2
cp7	Mismatching data semantics in a message	5.2.1.2
cp8	Mismatching state maintenance assumptions	Not addressed
cp9	Mismatching state scope assumptions	5.2.1.5
cp10	Mismatching failure mode assumptions	5.2.1.7
cp11	Mismatching connector creation/destruction assumptions	5.2.1.8
cp12	Connection to a non public web service port	5.2.1.9
cp13	Connected ports must share transport and encoding protocols	5.2.1.9
	Component to environment scope	
cc1	Concurrent calls to a no queuing and non-reentrant port	5.2.2.7
cc2	Mismatching conversations	5.2.2.2 and 5.2.2.3
cc3	Partially matching conversations	5.2.2.2 and 5.2.2.3
cc4	No component has an active thread of control	Not addressed
cc5	Concurrent threads in a single thread only component	Not addressed
cc6	Concurrent threads in a non-reentrant port	5.2.2.7
cc7	Mismatching process distribution assumpations	Not addressed
	Type checking	
ct1	Non web service compliant connector	5.2.4
ct2	Non web service compliant component	5.2.4
ct3	Ports must be well defined	5.2.1.9
ct4	Components must have correct port types	5.2.3.1
ct5	Components must be well defined	5.2.3.1
ct6	Connectors must be well defined	5.2.3.3
ct7	Roles must be well defined	Roles have no properties so no well defined checks are performed

Table 5.6: The sections in which each mismatch type is addressed.

Chapter 6

Case Study and Evaluation

Previous chapters have described the derivations of both minimal and enhanced web service architectural styles; this chapter will demonstrate their effectiveness in representing a system and detecting mismatches.

The chapter is in three parts. The first part presents a case study used to demonstrate a range of mismatches detectable by the minimal style. The second section moves on to demonstrate some of the mismatch detection capabilities of the enhanced style. In this section the style is used to represent a system from the literature showing that it can both be used to detect the mismatches discussed and the confirm their removal from the resulting corrected system. The final section looks at the enhanced style from a number of different view points relating to the accuracy and effectiveness of its analysis and the results presented.

6.1 ACME Studio Graphical View Key

Throughout this chapter screen shots of the graphical view in ACME Studio will be used to illustrate the system being discussed and how the mismatches are initially indicated to the user. To aid with the understanding of these figures, a key relating the element types and their graphical representations is shown in Table 6.1.

6.2 Case Study to Evaluate the Minimal Style

This first part of the evaluation of this work shows a case study developed to demonstrate the capabilities the minimal web service architectural style. The scenario covers an in-car satellite navigation system based upon existing services with some extra functionality added. Fragments of the system model, defined in ACME, will be presented in this section to illustrate the key points while the full ACME description may be found in Appendix C.

The service being developed consists of two separate software components: the satellite navigation

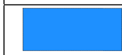
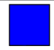
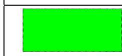
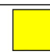
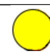
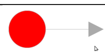
Image	*Type*	*Image*	*Type*
Components		Ports	
	`CompTWSClient`		`PortTWSClientSingle`
	`CompTWSService`		`PortTWSClientUnicast`
	`CompTWSIntermediary`		`PortTWSServiceSingle`
	`CompTWSAnalysiControl`		`PortTWSServiceUnicast`
Connectors		Misc.	
	`ConnTWSCommon`		ACME Mismatch Warning
	`ConnTWSCooperative`	*Sample text*	Diagram Annotation
	`ConnTWSStubborn`		

Table 6.1: A key to the elements in the graphical view of the style in ACME Studio.

provider (SNP), which is centralised at some data centre and an in-car navigation unit (NU). The NU has the usual functionality of selecting a route from the current location to a specified address, but it can also delegate route calculation back to the systems at SNP via web service connections over a General Packet Radio Service (GPRS[1]) connection. The routes calculated can then take into account the latest traffic reports and road works, leading to a potentially much better route choice. The central SNP systems can also update the route provided to individual NUs if there is a relevant traffic situation change. This is done by querying the current location of the vehicle and sending a new route plan if appropriate.

A second addition to the normal satellite navigation functionality is that the SNP will contact and direct recovery services to the vehicle if a breakdown is signalled. To enhance the service provided, the NU can obtain some diagnostic information from the vehicle's engine management unit (if available) so the recovery service can respond to the situation in the most appropriate way. The information is obtained from the engine management unit using web service protocols and is assumed to consist of raw sensors' information. Thus, we also include a service provided by the car manufacturers whereby they will decode and collate the sensors' data and return a plain text diagnostic. The diagnostic, vehicle location and passenger status is passed to a number of recovery services, which return their assistance offers, consisting of estimated time of arrival (ETA), cost and details such as if they intend to attempt to repair on site or just to tow away. The user can then select which of the service offers to accept. Additionally, the recovery services may need to alter their ETA as a result of other breakdowns that have a higher priority, such as a lone female driver at night, in which case the new details of the recovery can be sent to the NU.

[1] `http://gsmworld.com/technology/gprs.htm` or for the specification detail see `http://www.3gpp.org/ftp/Specs/html-info/0260.htm`.

Figure 6.1 shows the initial proposed architecture of this system consisting of components to represent SNP and NU that are being developed, as well as existing external services: two recovery services (RS1 & RS2), two car manufacturers (CM1 & CM2) and a selection of their engines with their corresponding management units (CM1E1, CM1E2 & CM2E1). These have been described using the minimal web service architectural style within the ACME Studio environment.

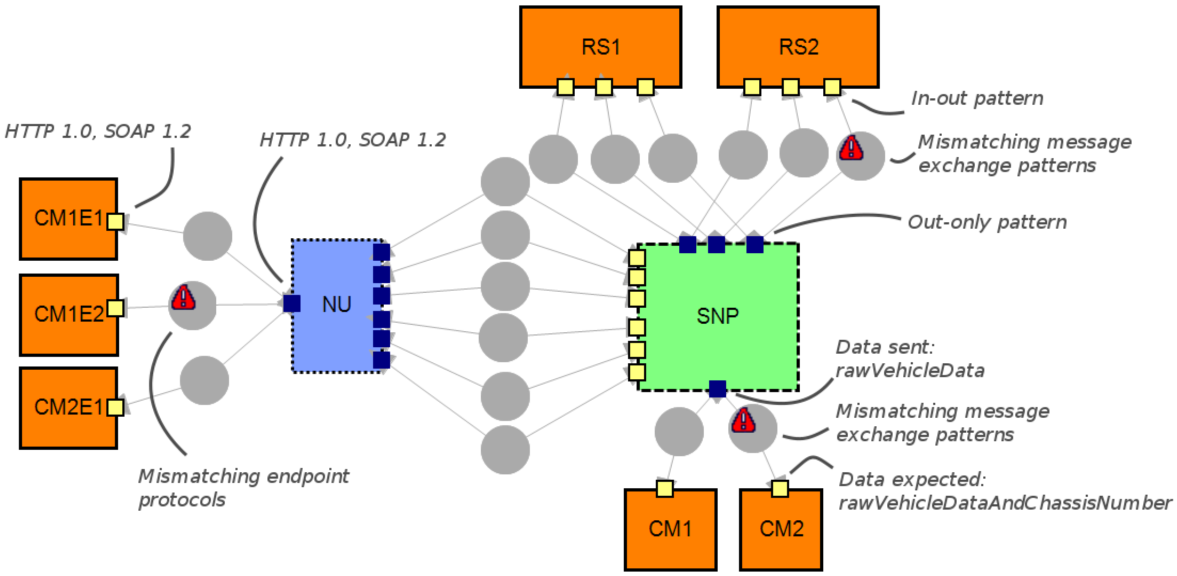


Figure 6.1: The initial system architecture with warning triangles showing where mismatches have been detected.

ACME Studio has placed warning triangles on three of the connectors in the architecture. These warning triangles are overlaid on components or connectors to indicate that one or more constraints on them are not met. In this case that means that an architectural mismatch has been detected and is localised around that connector. A triangle does not indicate what the nature of the mismatch is for that one must select the connector in question and note which of the rules are reported as failed. Figure 6.2 shows this view for the connector between NU and CM1E2. The rule indicates that there is no matching pair of endpoint protocols shared between the two ports as shown in the following two fragments from the architecture description, the first being from the port on NU and the second being from the port on CM1E2.

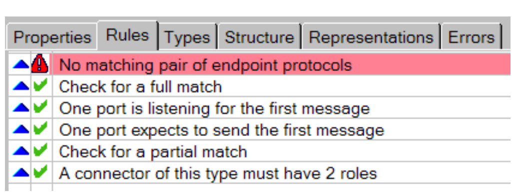


Figure 6.2: The rule summary for the connector between NU and CM1E2

```
// extract from the original NU port description
Property EndPointList : EndPoints = {[
    Transport = HTTP1_0;
    Encoding = SOAP1_1 ]};
```

```
// extract from the CM1E2 port description
Property EndPointList : EndPoints = {[
    Transport = HTTP1_0;
    Encoding = SOAP1_2 ]};
```

This is corrected by changing the SOAP processor used by the NU to one which supports both SOAP 1.1 and SOAP 1.2, which is described by altering the port description to be as follows.

```
// extract from the updated NU port description
Property EndPointList : EndPoints = {[
    Transport = HTTP1_0;
    Encoding = SOAP1_1 ], [
    Transport = HTTP1_0;
    Encoding = SOAP1_2 ]};
```

The second warning is found on a connector between the SNP and RS2, examining the rules reveals that the mismatch relates to the messages exchanged between the ports, Figure 6.3. From the descriptions we can learn that while the port on RS2 expects a request response message exchange pattern, the port on SNP is using a one way (notification) pattern, shown in Figure 6.4. This is so RS2 can get a confirmation that its services are still required if it has to change details of a previously accepted offer.

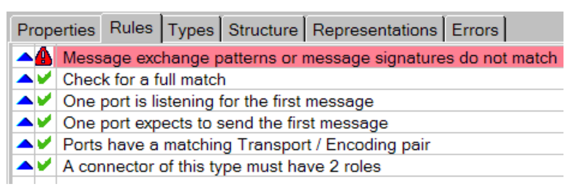


Figure 6.3: The rule summary for the connector between SNP and RS2

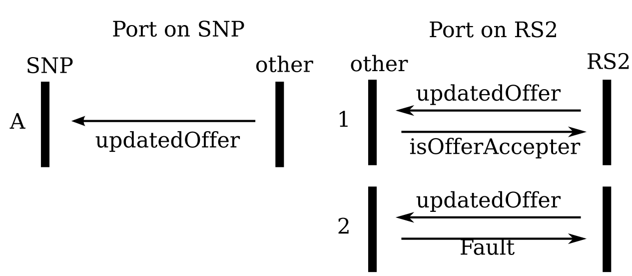


Figure 6.4: The mismatching message exchange patterns between SNP and RS2

```
// extract from the original SNP port description
Property MessageExchangePatterns : messagePatterns = {< [
  ST = "updateOffer";
  DT = "out" ] >};
```

```
// extract from the original RS2 port description
Property MessageExchangePatterns : messagePatterns = {< [
  ST = "updateOffer";
  DT = "out" ], [
  ST = "isUpdateAccepted";
  DT = "in" ] >, < [
  ST = "updateOffer";
  DT = "out" ], [
  ST = "fault";
  DT = "in" ] >};
```

To correctly interoperate with RS2 then it is necessary to add a new port to SNP which follows the expected interaction[2]. Then for completeness the interface between NU and SNP is altered such that the user can make the decision whether to accept the new offer or not.

The final warning exists on the connector between SNP and CM2. The rules summary for this connector shows that the same rule failed as for the previous connector, however, examining the message exchange patterns shows that they are both of the request response type. So in this situation the tokens representing the data included in each message must be considered to find where the problem lies. CM2 requires an additional data item to be sent before it can respond with a diagnostic report, this is the vehicle chassis number that is not included in the raw sensor data. To avoid this mismatch another client port is added to SNP which has the same message exchange pattern as the original but also includes this extra information.

```
// extract from the original SNP port description
Property MessageExchangePatterns : messagePatterns = {< [
  ST = "rawVehicleData";
  DT = "out" ], [
  ST = "diagnosticInformation";
  DT = "in" ] >, < [
  ST = "rawVehicleData";
  DT = "out" ], [
  ST = "fault";
  DT = "in" ] >};
```

```
// extract from the original CM2 port description
Property MessageExchangePatterns : messagePatterns = {< [
  ST = "rawVehicleDataAndChassisNumber";
  DT = "out" ], [
  ST = "diagnosticInformation";
  DT = "in" ] >, < [
  ST = "rawVehicleDataAndChassisNumber";
  DT = "out" ], [
  ST = "fault";
  DT = "in" ] >};
```

With these corrections made, the final architecture (shown in Figure 6.5) has no mismatches detected according to this architectural style. So actual development of the software components

[2]i.e. the description of the new SNP port message exchange patterns property becomes identical to that of the RS2 port.

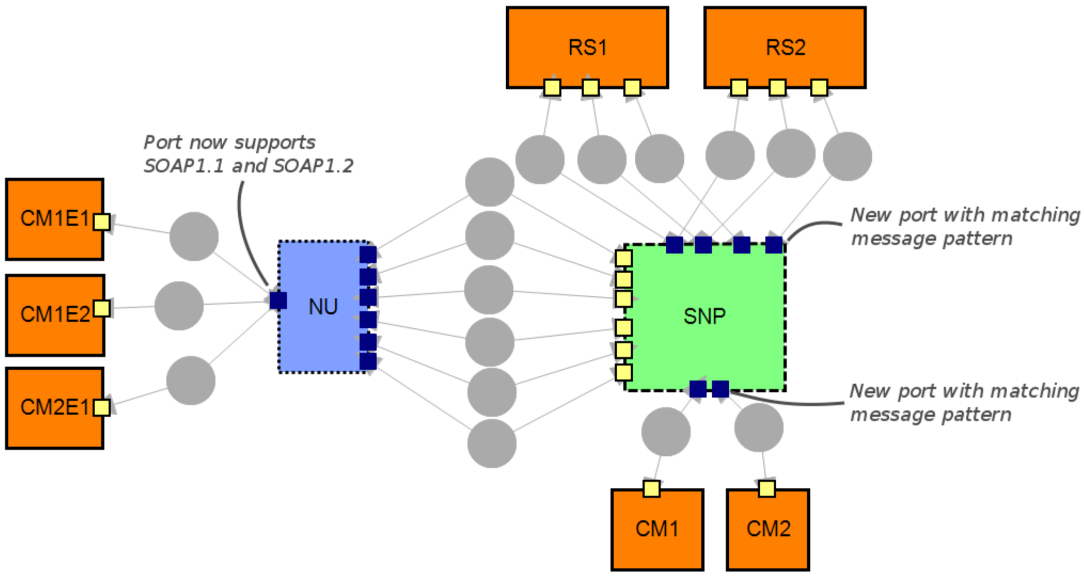


Figure 6.5: The final architecture of the envisaged system.

NU and SNP could now begin with greater confidence of success.

6.2.1 Section Summary

This section demonstrated that the style is able to detect mismatches in the example system described, but what can be said about its applicability to other systems? The question to answer here is, would the analyses included in the style be able to detect the same mismatch types in any other web service system? The answer to this question lies in both the scope of the analysis rules and the nature of the properties they act upon.

The rules all have very restricted scope, they are all limited to either a single port, a single connector or a single component, the exceptions to this are the two system wide type checking rules. Those rules that have the scope of a single component or port are used to confirm that the element in question is well defined either by including all the required properties or by containing the correct sub-elements, for example a client component only contains client type ports. This first type of rule does not consider the other components in the system at all and so cannot be affected by them. The rules in the connectors consider the properties of the ports at both ends and so long as both of those ports have the properties required by the style, the analysis represented by the rules should work correctly. The connector rules do assume that only point-to-point connections exist and therefore each connector has exactly two roles and that each is attached to a single port. This assumption is codified in a single rule asserting that a connector has two roles, so a connector that would invalidate the assumptions of the analysis rules would be flagged to the user. Finally the two rules with a scope wider than a single element simply assert that each component or connector in the system satisfies one of the types defined in the style.

The second factor to consider is the nature of the properties feeding the analyses. In the case of the minimal style all properties included in the analyses are static in nature, being described by the architect when the model is created, there are no analyses that are based upon the emergent behaviour of the composed system. The argued answer to the earlier question then is that there is a high degree of confidence that the analysis in the minimal architectural style would be effective at discovering mismatches in any system constructed using it.

The next sections show that the enhanced style is capable of detecting the mismatches derived in Chapters 4 and 5. It is also capable of detecting all mismatches included in the minimal style but demonstration of this is not included for sake of brevity.

6.3 Case Studies to Evaluate the Enhanced Style

6.3.1 Car Parking

The first scenario used to demonstrate the enhanced style is based upon the work of Cavallaro and Di Nitto [CN08]. Their work is complementary to this thesis as it assumes a situation where one or more mismatches have been detected in a system. The approach they illustrate allows the adaptation of semantically equivalent services so they exhibit the same interface protocols through the use of a mediator framework and scripts.

The complete ACME descriptions of both the initial and final configurations discussed in this section may be found in Appendices E.1.1 and E.1.2 respectively.

The approach is outlined using the example of a pair of car park pre-payment services, `BookingPaymentCC` and `SpaceCCBuy`[3], along with a client application that is required to connect to both services. The name `CPClient` will be applied to the client in this work. Abstractly the client expects to be able to log-in to a car park service, make a payment to reserve a space and then log out again. Tables 6.2 and 6.3 show the interfaces provided by both services and while the names for the data change slightly and the data is not formally described in any way, it is possible to see by inspecting the parameters columns that the same information is required by both services.

Further inspection of the interfaces reveals that a mismatch exists in the form of a different sequence of messages expected when making a payment. In the `BookingPaymentCC` protocol there is a single solicit-response message pattern containing all the details required for the transaction while in the `SpaceCCBuy` protocol, the card and ownership details are transferred in one solicit-response exchange and then the amount to be paid is conveyed separately. This is essentially the only difference between the two services, both of which share a similar, linear, process flow through

[3]The original paper used the service names `BookingPaymentCC` and `BookingCCPayment`, these sound quite close to each so to reduce confusion `BookingCCPayment` was replaced with `SpaceCCBuy` for this work. Also `BookingPaymentCC` is truncated to `BookPayCC` in the models to reduce space.

Operation name	*Parameters*	*Return value*
setupConf	String:userName String:password	boolean:success
paymentCC	String:owner String:CCNumber float:amount Date:expirationDate	boolean:success
logout		boolean:success

Table 6.2: The interface offered by the BookingPaymentCC service

Operation name	*Parameters*	*Return value*
setupConf	String:user String:password	boolean:success
checkCreditCard	String:owner String:cardNumber Date:expDate	boolean:success
payByCC	float:amount	boolean:success
logout		boolean:success

Table 6.3: The interface offered by the SpaceCCBuy service

the available messages that may be exchanged as shown in the form of simple state machines in Figure 6.6.

To satisfy the requirements of the style a number of assumptions were made about the components that were not made explicit in the paper as they were out of scope. The actual values chosen are not of great importance as they do not impact on the detection of the mismatches in the scenario at all. The one slight exception to this stems from the very optimistic view taken by Cavallaro and Di Nitto about the success of each message exchange, specifically the protocols ignore the possibility of any port invocation returning a fault message. This is merely noted as a slight oddity in this scenario as it seems likely that, for example, a fault message returned from the checkCreditCard in the `BookingPaymentCC` protocol should not then lead to a state where the protocol is considered 'ready for payment'. However as it is possible to imagine both protocols with a more realistic treatment of fault messages and at the same time both patterns remaining semantically equivalent to each other, the original protocols described by Cavallaro and Di Nitto are used in the ACME architecture model.

6.3.1.1 Initial Configuration and Mismatches

The client proposed by Cavallaro and Di Nitto is based upon the interface exposed by the `BookingPaymentCC` service, so there is an implicit mapping between the ports of the client and that service. The result is that connecting the client to that service is trivial and results, as expected, in no mismatches.

The `SpaceCCBuy` service has a different interface and so while there is an obvious match between

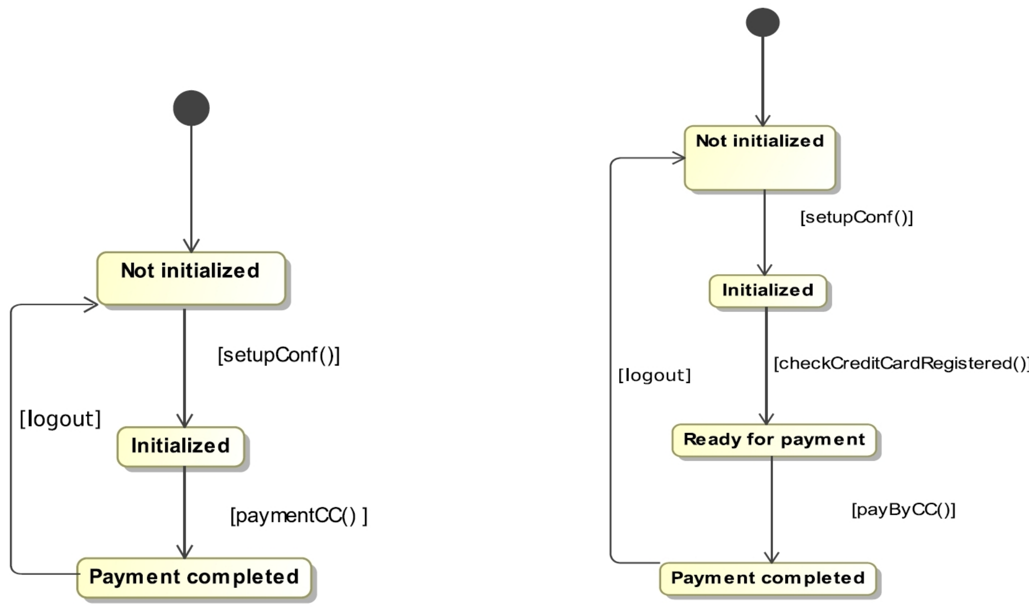


Figure 6.6: The protocols expected by the services in the car park scenario. On the left is BookPayCC protocol, and on the right the SpaceCCBuy protocol

the `login` and `logout` ports of the client and service, it is not apparent how to connect the remaining ports. There are, however, two options that can be tried.

- `CPClient.paymentCC` connects to `SpaceCCBuy.checkCreditCard`; or
- `CPClient.paymentCC` connects to `SpaceCCBuy.payByCC`.

Both of these options were constructed and the resulting models in ACME Studio are shown in Figure 6.7. Both models result in mismatches being detected as indicated by the presence of the red warning triangles, one on the component and another on the connector described above.

Considering the `CPCClient` component warning triangle first and consulting the rules view in ACME Studio informs us that the mismatch indicated is a commission partial match. This is one of the external analysis rules developed as part of the style and additional information regarding the details of the mismatch is available from the text file output by the analysis. For this rule type, the output describes the traces returned from the FDR model checker that lead to the sending of the additional, unexpected message. The analysis output generated from the initial configuration is as follows:

```
CPClient attempted to send unexpected messages (commission events) in 1 traces.
Commission trace number 1
CPClient_setupConf_sendReq_SpaceCCBuy SpaceCCBuy_login_sendReq_CPClient
SpaceCCBuy_login_getFault_CPClient CPClient_setupConf_getFault_SpaceCCBuy
CPClient_PaymentCC_sendReq_SpaceCCBuy SpaceCCBuy_checkCreditCard_sendReq_CPClient
```

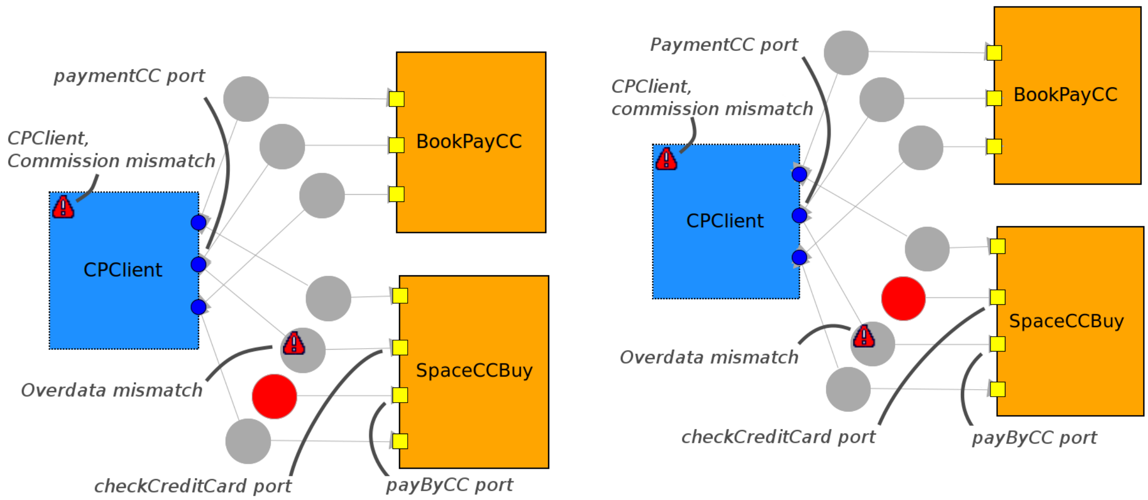


Figure 6.7: The alternative initial configurations of the car park system. On the left with `CPClient.paymentCC` connected to `SpaceCCBuy.checkCreditCard` and on the right `CPClient.paymentCC` connected to `SpaceCCBuy.payByCC`. A stubborn connector is used to indicate that there is no known connection to one of the `SpaceCCBuy` ports in each case

```
SpaceCCBuy_checkCreditCard_getFault_CPClient CPClient_PaymentCC_getFault_SpaceCCBuy
CPClient_logout_sendReq_SpaceCCBuy
```

While the alternate configuration results in the following analysis output:

```
CPClient attempted to send unexpected messages (commission events) in 1 traces.
Commission trace number 1
CPClient_setupConf_sendReq_SpaceCCBuy SpaceCCBuy_login_sendReq_CPClient
SpaceCCBuy_login_getFault_CPClient CPClient_setupConf_getFault_SpaceCCBuy
CPClient_PaymentCC_sendReq_SpaceCCBuy
```

Both outputs inform the architect that the client component is attempting to send a message that is not expected by its environment as the final message in both traces emanates from the client component.

Proper use of the naming scheme for messages described in the style assists greatly in interpreting these traces. A message should always have a name that starts with the component ID, followed by the port ID and then finally the identifier of that message within the port. The external analysis then appends this given name with the ID of the component it will be sent to or received from. The first message shown in the trace above is named `CPClient_setupConf_sendReq_SpaceCCBuy`. This means the message was defined in the `CPClient`, in the `setupConf` port and was called `sendReq`, the name implies the message was sent from this port and its target is a port on the component `SpaceCCBuy`.

With this and the knowledge that the final message in the trace is the one that was sent unexpectedly, we can see that in the first configuration the client attempts to interact with the `logout` port of the service after interacting with the `checkCreditCard` port. An inspection of the protocol shows us that this is not allowed as the service expects an interaction on the `payByCC` port before a logout is allowed. In the second configuration the client attempts to send a message to the `payByCC` port without interacting with the `checkCreditCard` port, again an inspection of the protocol for this service shows that this is not allowed.

This confirms that the client is not directly compatible with the `SpaceCCBuy` service in terms of the number of messages exchanged and that there is some mediation required.

The second warning triangle reports a mismatch on the connector between the `CPClient.paymentCC` port and the `SpaceCCBuy.checkCreditCard` or `SpaceCCBuy.payByCC` port depending on which configuration is being observed. Examining the rule view for the faulty connector in both variants of the system reveals a "message over data" mismatch in the first message in the sequence. This rule is implemented using the external analysis facility and so allows the output of additional descriptive information, in this case the output reveals the IDs of the data in the sent message that are not required by the recipient.

The initial configuration gives the following output:

```
The following data was sent but is not expected:  amount
```

And the alternative configuration gives this output:

```
The following data was sent but is not expected:  owner
The following data was sent but is not expected:  CCNumber
The following data was sent but is not expected:  expirationDate
```

From a mediation point of view the results tell us two things:

- the lack of any mismatches of the "under data" type means that the client is sending all the data required by the service; and
- it is possible to describe which items of data should be filtered out of the messages for each port from the datum IDs listed.

Addition of Adaptation Framework

In the paper, service adaption takes place between the client and service instances with a run-time choice of which service to employ. In the ACME model a new intermediary type component is added to represent the SCENE adaption framework Cavallaro and Di Nitto reference. The adaptation framework assumes there is an abstract interface, which in this case is identical to that provided by the `BookingPaymentCC` component. To reflect this, the ACME model of the `SCENE` component is

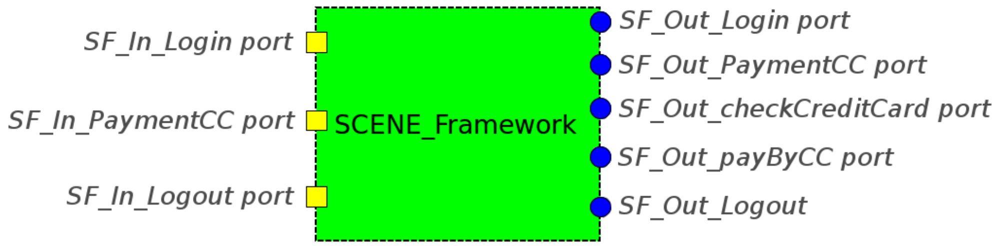


Figure 6.8: The `SCENE_Framework` component, with its service ports on the left and the client ports on the right

initially populated with a set of ports and properties that are consistent with the `BookingPaymentCC` component.

The actual adaptation in the framework is represented by a number of mappings though only two of these impact this model of the system, these are the operation mapping and parameter mapping. Attending first to the more coarse grained operation mapping we see that the `paymentCC` operation in the abstract interface is replaced by the sequential invocation of `checkCreditCard` and then `payByCC` operations when utilising the `SpaceCCBuy` service. As we are aware that these operations not only have different names but both contain a subset of the parameters `paymentCC` operation, two new ports were added, both populated, for the time being, with the same properties as `paymentCC`. The graphical form of this component is shown in Figure 6.8.

Protocol Adaptation

It was then necessary to adjust the process names in the port CSP templates and also make a change to the central component CSP to allow either service to be selected at run-time and also to ensure that the correct choreography for each is observed.

Starting with the central CSP, we define a thread process that is initially willing to accept a request from the client on the `In_login` port. Upon receiving this request, the process breaks out from that port's CSP template and is forwarded to the `Out_login` port. This `Out_login` port is where the choice is made about which service component to interact with and so is made the choice maker for the choice group 'services', which includes all the client ports on this component. The outcome points of the CSP template in the `Out_login` port are pointed toward the appropriate points on the `In_login` template such that the identical response message is returned to the client. Finally for the login ports, both outcome points on the `In_login` template are directed towards the `In_paymentCC` port, which is the next in the choreography. The CSP templates for both of these ports are recounted below.

$SF\_In\_login \mathrel{\widehat=} SF\_In\_login\_getReq \rightarrow SF\_Out\_login$
$SF\_In\_login\_p1 \mathrel{\widehat=} SF\_In\_login\_p2 \;\Box\; SF\_In\_login\_p3$
$SF\_In\_login\_p2 \mathrel{\widehat=} SF\_In\_login\_sendRes \rightarrow SF\_In\_login\_OK$
$SF\_In\_login\_p3 \mathrel{\widehat=} SF\_In\_login\_sendFault \rightarrow SF\_In\_login\_FAULT$
$SF\_In\_login\_OK \mathrel{\widehat=} SF\_In\_PaymentCC$
$SF\_In\_login\_FAULT \mathrel{\widehat=} SF\_InPaymentCC$

$SF\_Out\_login \mathrel{\widehat=} SF\_Out\_login\_sendReq \rightarrow SF\_Out\_login\_p1$
$SF\_Out\_login\_p1 \mathrel{\widehat=} SF\_Out\_login\_p2 \;\Box\; SF\_Out\_login\_p3$
$SF\_Out\_login\_p2 \mathrel{\widehat=} SF\_Out\_login\_getRes \rightarrow SF\_Out\_login\_OK$
$SF\_Out\_login\_p3 \mathrel{\widehat=} SF\_Out\_login\_getFault \rightarrow SF\_Out\_login\_FAULT$
$SF\_Out\_login\_OK \mathrel{\widehat=} SF\_In\_login\_p2$
$SF\_Out\_login\_FAULT \mathrel{\widehat=} SF\_In\_login\_p3$

When the login process is complete the interaction moves to the next step where the client will pay for a parking space. This process is initiated by the port `SF_In_PaymentCC` receiving the request message from the client application, this message is represented by `SF_In_PaymentCC_getReq` in the CSP below.

$SF\_In\_PaymentCC \mathrel{\widehat=} SF\_In\_PaymentCC\_getReq \rightarrow SF\_Process\_Branch$
$SF\_In\_PaymentCC\_p1 \mathrel{\widehat=} SF\_In\_PaymentCC\_p2 \;\Box\; SF\_In\_PaymentCC\_p3$
$SF\_In\_PaymentCC\_p2 \mathrel{\widehat=} SF\_In\_PaymentCC\_sendRes \rightarrow SF\_In\_PaymentCC\_OK$
$SF\_In\_PaymentCC\_p3 \mathrel{\widehat=} SF\_In\_PaymentCC\_sendFault \rightarrow SF\_In\_PaymentCC\_FAULT$
$SF\_In\_PaymentCC\_OK \mathrel{\widehat=} SF\_In\_logout$
$SF\_In\_PaymentCC\_FAULT \mathrel{\widehat=} SF\_In\_logout$

It is after the request message has been received that the selection of the correct protocol for the chosen car park service takes place. This selection is achieved via two mechanisms. The first mechanism is a branching process added to the component's central CSP. This process, which is called immediately after the request message is received above, allows the process flow to branch in either of two directions, one direction meeting the `BookPayCC` protocol and the other meeting the `SpaceBuyCC` protocol. The choice of direction is dictated by the second mechanism, specifically that both ports referenced in the branching process are part of the 'services' choice group. The result is that the process can only proceed down the path representing the correct protocol for the service chosen during the earlier login step.

$SF\_Process\_Branch \mathrel{\widehat=} SF\_Out\_paymentCC \;\Box\; SF\_Out\_checkCreditCard$

In the case that the first branch is taken, the process moves to the CSP included in the

SF_Out_PaymentCC port, shown below. This port adheres to the payment part of the BookPayCC protocol by sending a paymentCC message to the required server and expects a single message in return. The process flow is then redirected to the SF_In_PaymentCC port at either SF_In_PaymentCC_p2 if a normal response message was received, or at SF_In_PaymentCC_p3 if the response indicated a fault.

$$SF\_Out\_PaymentCC \mathrel{\widehat{=}} SF\_Out\_PaymentCC\_sendReq \rightarrow SF\_Out\_PaymentCC\_p1$$
$$SF\_Out\_PaymentCC\_p1 \mathrel{\widehat{=}} SF\_Out\_PaymentCC\_p2 \mathrel{\Box} SF\_Out\_PaymentCC\_p3$$
$$SF\_Out\_PaymentCC\_p2 \mathrel{\widehat{=}} SF\_Out\_PaymentCC\_getRes \rightarrow SF\_Out\_PaymentCC\_OK$$
$$SF\_Out\_PaymentCC\_p3 \mathrel{\widehat{=}} SF\_Out\_PaymentCC\_getFault \rightarrow SF\_Out\_PaymentCC\_FAULT$$
$$SF\_Out\_PaymentCC\_OK \mathrel{\widehat{=}} SF\_In\_PaymentCC\_p2$$
$$SF\_Out\_PaymentCC\_FAULT \mathrel{\widehat{=}} SF\_In\_Payment\_p3$$

If the other branch was taken then the process moves to the CSP included in the SF_Out_checkCreditCard port, shown below. The port adheres to the first step when making a payment using the SpaceCCBuy protocol. It sends the checkCreditCard message expected by the protocol and waits for a message in response. When the response is received the process is then directed to the SF_Out_payByCC port. This port sends the payByCC message expected next in the protocol and then waits for the message response. As with the process description presented above, the process flow is then returned to the SF_In_PaymentCC port at the correct point to indicate whether a normal response or a fault message was received.

$$SF\_Out\_checkCreditCard \mathrel{\widehat{=}} SF\_Out\_checkCreditCard\_sendReq \rightarrow SF\_Out\_checkCreditCard\_p1$$
$$SF\_Out\_checkCreditCard\_p1 \mathrel{\widehat{=}} SF\_Out\_checkCreditCard\_p2 \mathrel{\Box} SF\_Out\_checkCreditCard\_p3$$
$$SF\_Out\_checkCreditCard\_p2 \mathrel{\widehat{=}} SF\_Out\_checkCreditCard\_getRes \rightarrow SF\_Out\_checkCreditCard\_OK$$
$$SF\_Out\_checkCreditCard\_p3 \mathrel{\widehat{=}} SF\_Out\_checkCreditCard\_getFault \rightarrow SF\_Out\_checkCreditCard\_FAULT$$
$$SF\_Out\_checkCreditCard\_OK \mathrel{\widehat{=}} SF\_Out\_payByCC$$
$$SF\_Out\_checkCreditCard\_FAULT \mathrel{\widehat{=}} SF\_Out\_payByCC$$

$$SF\_Out\_payByCC \mathrel{\widehat{=}} SF\_Out\_payByCC\_sendReq \rightarrow SF\_Out\_payByCC\_p1$$
$$SF\_Out\_payByCC\_p1 \mathrel{\widehat{=}} SF\_Out\_payByCC\_p2 \mathrel{\Box} SF\_Out\_payByCC\_p3$$
$$SF\_Out\_payByCC\_p2 \mathrel{\widehat{=}} SF\_Out\_payByCC\_getRes \rightarrow SF\_Out\_payByCC\_OK$$
$$SF\_Out\_payByCC\_p3 \mathrel{\widehat{=}} SF\_Out\_payByCC\_getFault \rightarrow SF\_Out\_payByCC\_FAULT$$
$$SF\_Out\_payByCC\_OK \mathrel{\widehat{=}} SF\_In\_PaymentCC\_p2$$
$$SF\_Out\_payByCC\_FAULT \mathrel{\widehat{=}} SF\_In\_PaymentCC\_p3$$

Regardless of which protocol was observed for payment, the process is now directed to the SF_In_logout port. This port also contains a breakout to forward the request to the SF_Out_logout port. This latter port is also part of the choice group as the logout request should be directed toward the service interacted with. Again the received response message causes the process to move to the appropriate point on the SF_In_logout to allow the correct message to be returned to the client.

Both outcome points of the CSP then direct the process back to the starting point of the whole protocol.

$$
\begin{aligned}
&SF\_In\_logout \mathrel{\widehat=} SF\_In\_logout\_getReq \rightarrow SF\_Out\_logout\\
&SF\_In\_logout\_p1 \mathrel{\widehat=} SF\_In\_logout\_p2 \;\Box\; SF\_In\_logout\_p3\\
&SF\_In\_logout\_p2 \mathrel{\widehat=} SF\_In\_logout\_sendRes \rightarrow SF\_In\_logout\_OK\\
&SF\_In\_logout\_p3 \mathrel{\widehat=} SF\_In\_logout\_sendFault \rightarrow SF\_In\_logout\_FAULT\\
&SF\_In\_logout\_OK \mathrel{\widehat=} SF\_Thread\\
&SF\_In\_logout\_FAULT \mathrel{\widehat=} SF\_Thread
\end{aligned}
$$

$$
\begin{aligned}
&SF\_Out\_logout \mathrel{\widehat=} SF\_Out\_logout\_sendReq \rightarrow SF\_Out\_logout\_p1\\
&SF\_Out\_logout\_p1 \mathrel{\widehat=} SF\_Out\_logout\_p2 \;\Box\; SF\_Out\_logout\_p3\\
&SF\_Out\_logout\_p2 \mathrel{\widehat=} SF\_Out\_logout\_getRes \rightarrow SF\_Out\_logout\_OK\\
&SF\_Out\_logout\_p3 \mathrel{\widehat=} SF\_Out\_logout\_getFault \rightarrow SF\_Out\_logout\_FAULT\\
&SF\_Out\_logout\_OK \mathrel{\widehat=} SF\_In\_logout\_p2\\
&SF\_Out\_logout\_FAULT \mathrel{\widehat=} SF\_In\_logout\_p3
\end{aligned}
$$

Message Data Adaption

With the adaptation of the protocol now correct in terms of the number and direction of messages we now attend to the parameters mapping to correct the data passed.

There are two aspects to this part, 'what data' is included in each message and 'what name' each item is given. As discussed in Chapter 5, the actual names assigned to parameters are not considered to be significant as these are just identifiers that could be altered without affecting the system behaviour at all. So for our purposes we just consider the semantics of the data included in each message.

The `Out_checkCreditCard` and `Out_payByCC` ports were initially populated with the properties from the `Out_paymentCC` port. We know from Tables 6.2 and 6.3 that the first two ports each contain a subset of the data exchanged by the original port, so the adaptation takes the form of deleting the unrequired data from each message. The details of which data in each message is unrequired can also be found in the output of the "over data" rule.

Figure 6.9 contains the initial `messages` data structure that both ports inherited followed by the final, reduced versions that the `Out_checkCreditCard` and `Out_payByCC` contain respectively.

There is now a complete representation of the `SCENE_Framework` adaptation component and we can see from the graphical view in ACME Studio, Figure 6.10, that the adaptation is correct according to our mismatch model as there are no warning triangles present any longer.

6.3.1.2 Section Summary

This section showed that the enhanced style can be used to represent a system described in the literature and could have been used to both determine the mismatches to be corrected by the

```
Port Out_paymentCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
  ...
  Property Messages : TMessages = {
    [MessageId = "SCENE_Framework_Out_PaymentCC_sendReq";MessageData = {
      [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
      [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
      [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
      [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
    [MessageId = "SCENE_Framework_Out_PaymentCC_getRes";MessageData = {
      [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
    [MessageId = "SCENE_Framework_Out_PaymentCC_getFault";MessageData = {
      [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};
  ...
}

Port Out_checkCreditCard : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
  ...
  Property Messages : TMessages = {
    [MessageId = "SCENE_Framework_Out_checkCreditCard_sendReq";MessageData = {
      [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
      [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
      [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
    [MessageId = "SCENE_Framework_Out_checkCreditCard_getRes";MessageData = {
      [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
    [MessageId = "SCENE_Framework_Out_checkCreditCard_getFault";MessageData = {
      [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};
  ...
}

Port Out_payByCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
  ...
  Property Messages : TMessages = {
    [MessageId = "SCENE_Framework_Out_payByCC_sendReq";MessageData = {
      [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;]};],
    [MessageId = "SCENE_Framework_Out_payByCC_getRes";MessageData = {
      [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
    [MessageId = "SCENE_Framework_Out_payByCC_getFault";MessageData = {
      [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};
  ...
}
```

Figure 6.9: The `messages` properties of the three adapted ports in the `SCENE_Framework` component.

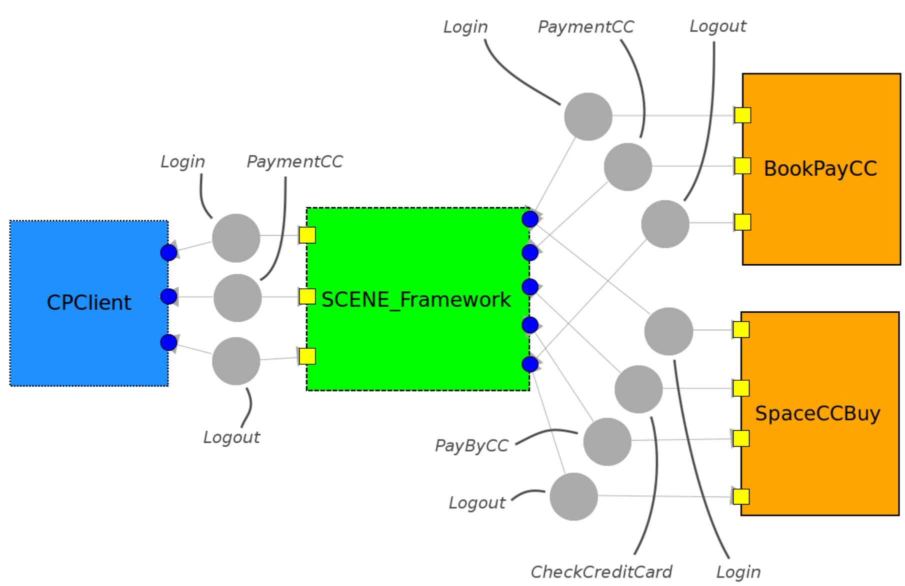


Figure 6.10: The final configuration of the car park scenario including the `SCENE_Framework` component. There are no mismatches reported

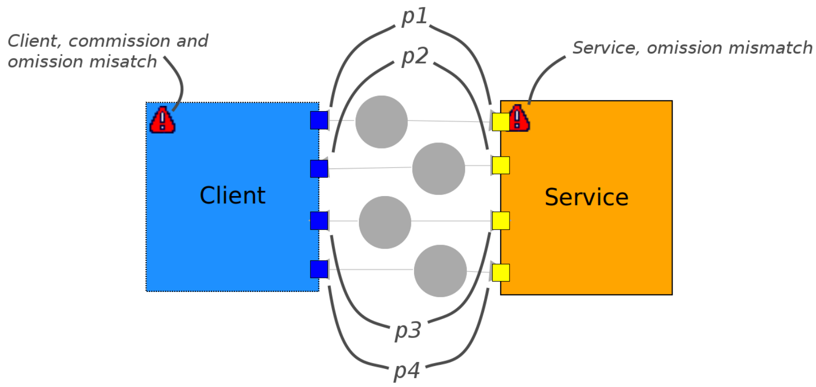


Figure 6.11: The configuration of the simple system used to demonstrate the omission check, with mismatches reported on both components

mediation component and confirm that they are no longer present in the final system.

This scenario does not, however, demonstrate some of the other interesting features of the enhanced style. The following sections describe simple systems in which these features can be shown.

6.3.2 Additional Tests : Omission

To demonstrate the analysis concerning omitted messages a simple system including two components each with four ports was constructed, Figure 6.11. Its ACME description may be found in Appendix E.2.1. The ports on both components are simply named p1...p4 and the components are designed to agree on all properties except the order in which they expect to interact on the ports. The client component expects to interact on its four ports as follows:

$$\begin{aligned} \mathit{Client} \;\widehat{=}\; & p1 \rightarrow p4 \rightarrow p2 \rightarrow \mathit{Client} \\ & \Box\; p3 \rightarrow \mathit{Client} \end{aligned}$$

While the service expects:

$$\begin{aligned} \mathit{Service} \;\widehat{=}\; & p1 \rightarrow p2 \rightarrow p3 \rightarrow \mathit{Service} \\ & \Box\; p4 \rightarrow \mathit{Service} \end{aligned}$$

Comparing the two conversations we can see that the two components only agree on one part of the conversation, the initial interaction on p1, they then make differing assumptions about the port that follows p1. The components also disagree on the port that represents the alternative conversation path to p1, p3 for the client and p4 for the service. From this we can say that four mismatches could be reported by the analysis:

Commission Mismatch:

Name	Rule
CommissionPartialMatch	EACommissionPartialMatch(self)
OmissionPartialMatch	EAOmissionPartialMatch(self)
CentralProcessDescribed	CentralProcessDescription != ""
ComponentInOurControlDomainDesc	ComponentInOurControlDomain == Yes OR ComponentInOurControlDomain == No
MsgDatumDescribed	EACentralDataStoreCorrect(self)

Figure 6.12: The ACME studio rule view for the Client component

Name	Rule
OmissionPartialMatch	EAOmissionPartialMatch(self)
CentralProcessDescribed	CentralProcessDescription != ""
ComponentInOurControlDomainDesc	ComponentInOurControlDomain == Yes OR ComponentInOurControlDomain == No
MsgDatumDescribed	EACentralDataStoreCorrect(self)

Figure 6.13: The ACME studio rule view for the Service component

- Client attempts to send a message to p4 after p1;
- Client attempts to send a message to p3 initially;

Omission Mismatch:

- Service expects a message on p2 after p1;
- Service can accept a message on p4 initially but Client cannot send it.

The actual analysis results, indicated in Figures 6.12 & 6.13, differ slightly from the expected results. The client component has a commission mismatch reported, as expected but also has an omission mismatch. The service component has an omission mismatch reported, as expected.

Opening up the analysis output files associated with the reported mismatches reveals the following details.

Client commission file

```
Client attempted to send unexpected messages (commission events) in 2 traces.

Commission trace number 1
Client_p3_sendReq_Service

Commission trace number 2
Client_p1_sendReq_Service Service_p1_getReq_Client
Service_p1_sendFault_Client Client_p1_getFault_Service
Client_p4_sendReq_Service
```

The file includes two traces ending in commission events. The first shows that the client will attempt to send `Client_p3_sendReq_Service` at the very start of the interaction and the second shows it will attempt to send `Client_p4_sendReq_Service` after a successful interaction on port p1. These are both consistent with the predicted results.

Client omission file

```
[Client_p1_sendReq_Service, Client_p1_getRes_Service,
Client_p4_sendReq_Service, Client_p4_getRes_Service]
```

This file shows that the client fails to receive the message `Client_p4_getRes_Service` after a successful interaction on port p1. This result was not predicted for the system and is in fact a false result as will be discussed shortly.

Service omission file

```
[Service_p4_getReq_Client]
```

The final output file shows that there was an omission mismatch relating to the service component, it does not receive the `Service_p4_getReq_Client` it is willing to receive. This omission was predicted but so was another, a message to port p2 after a successful interaction on port p1, which has not been reported by the analysis.

So, the analysis correctly identified 3 of the 4 mismatches predicted but it also flagged a mismatch that should not have been listed according to the predictions. We will consider the unpredicted omission linked to the client application first.

The trace found in the client omission file shows that the client component expects to interact on port p1 and then on port p4 but that it does not receive a response to the request it sends to port p4[4]. The earlier CSP specification shows that the service component expects to interact on port p2 after p1 and so is not willing to receive a message on port p4. This is backed up by the second trace found in the client commission file, where it can be seen that the client sending a request to port p4 is an unexpected event and would have resulted in the system deadlocking.

While it is true that the client will not receive a result from port p4, this is because the request it sends to that port is unexpected. The means that this omission event occurs after an earlier commission event. Recall from Section 5.2.2.3 that the omission analysis was designed to ignore potential false negative results by removing any omission event that occurs after a deadlock. In this case, as will now be explained, the "potential false negative" safeguard has failed.

Examining the trace for the omission mismatch we see that the client receives a normal response message (`Client_p1_getRes_Service`) from the service before attempting to interact with p4. In

[4]The output of the omission analysis shows a trace expected by the component that was not permitted by the system. The final event in this trace relates to a message that will not be sent in the current configuration.

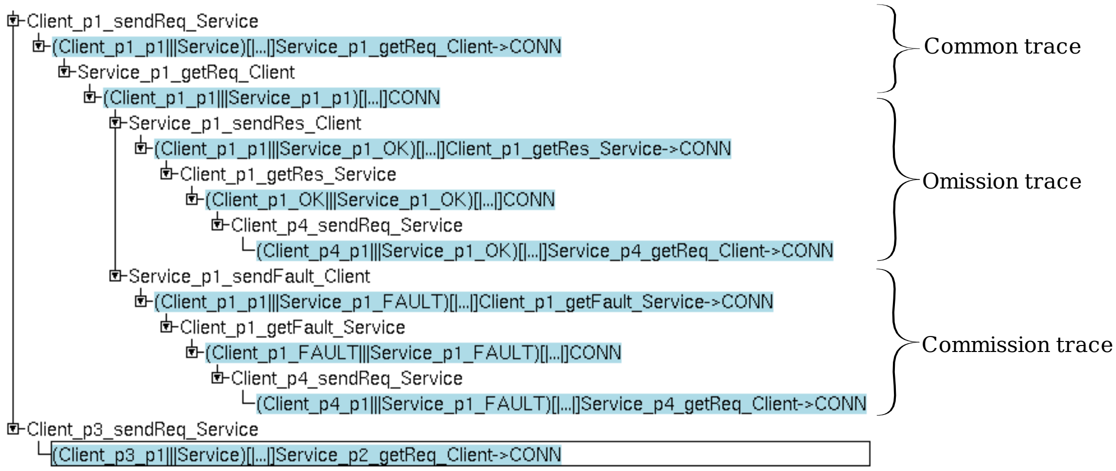


Figure 6.14: The ProBE output showing the traces possible in this system, with the trace returned by FDR for each mismatch highlighted

the earlier commission trace we see that the client receives a fault message (`Client_p1_getFault_Service`) from the service before it attempts to interact with p4. The omission analysis can only remove a potential false negative omission result if the trace leading to that omitted message contains the trace that leads to a commission event. In this case the traces are different and so the omission event is reported.

The root cause of this problem is that the FDR model checker does not return all traces leading to a deadlock failure. For example, in the second commission failure above, an examination of the model in the ProBE CSP animator tool[5] shows that there are two possible traces that can lead to the client attempting to send a message to p4. The lower path is that taken in the commission trace returned and involves the service returning a fault message to the client, the upper path is the path taken in the omission trace in which the service returns a normal response message. Without a complete set of deadlock traces it is not possible to guarantee the absence of potentially false negative results. Figure 6.14 shows the output from the ProBE tool confirming these traces.

Moving on to consider the missing omission analysis result, examining the service omission file reveals the single trace that was reported.

```
[Service_p4_getReq_Client]
```

This single trace matches the second of the two omission events predicted, while there is no indication of the first predicted omission event. Manually running the FDR tool on the CSP model generated to check for omission events returns only a single trace, the one shown above. However if we once again use the ProBE tool to explore the possible traces of the service component in isolation we find that they extend beyond those allowed by the system and therefore should result in further

[5]This tool makes it possible to explore all possible traces of a CSP model. It was obtained from Formal Systems Europe Ltd at `http://www.fsel.com/software.html`.

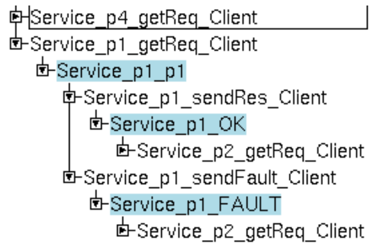


Figure 6.15: The ProBE output showing that the Service component has traces beyond those allowed by the system.

refinement failures. Figure 6.15 shows the ProBE results.

So again we see that FDR is not returning all possible failure traces and this is compromising the trust that can be placed in the results of the omission analysis. To balance this out it should be noted that at no point during the testing did FDR fail to report either a refinement failure or a deadlock failure when one existed in the system model, it just does not report them all. From an analysis point of view this means that potentially there may be both false positives and false negatives relating to the omission analysis. This risk can however be reduced[6] by first correcting any commission faults in the system and then tackling the omission faults.

6.3.3 Additional Tests: Cooperative Connector

To demonstrate the effect of using the `ConnTWSCooperative` connector type, a `CompTWSIntermediary` component acting as a simple service broker was constructed, the complete ACME description may be found in Appendix E.2.2. This component offers three service ports, s1...s3, for clients to connect to and also has three client ports, c1...c3, of its own that would connect to a chosen service provider.

The basic choreography expected by the component is described below using CSP and referencing the port IDs. Ports s1, s2, c1 and c2 are bound together in terms of choreography while s3 and c3 are not bound.

$$\begin{aligned} Broker \mathrel{\widehat{=}}\ & s1 \rightarrow c1 \rightarrow s2 \rightarrow c2 \rightarrow Broker \\ & \Box\ s3 \rightarrow Broker \\ & \Box\ c3 \rightarrow Broker \end{aligned}$$

The cooperative connector type was included in the style to indicate connections to unknown parts of the system. It assumes that those parts of the system work exactly as needed so that any mismatches reported on the model are found within the model rather than being a pessimistic assumption about unknown component properties.

[6]The risk of false results can only be reduced, not eliminated, at this point due to a fault in the one part of the external analysis code. This flaw is demonstrated and discussed in Section 6.3.4

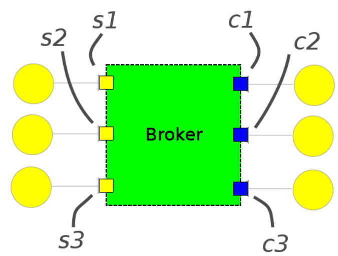


Figure 6.16: The configuration of the simple system used to demonstrate the cooperative connector

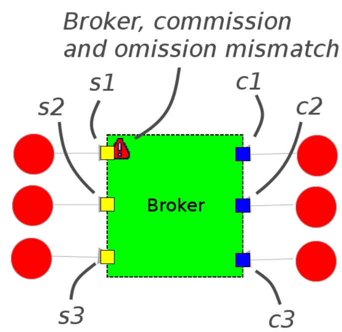


Figure 6.17: The configuration of the simple system used to demonstrate the stubborn connector

As expected then, and as shown in Figure 6.16, there are no mismatches reported in the system with the broker and all cooperative connectors.

6.3.4 Additional Tests: Stubborn Connector

The following test is a mirror of the previous, and demonstrates the effects of employing the `ConnTWSCooperative` connector type to the same broker component as before. The complete ACME description may be found in Appendix E.2.3. As this system contains many instances of both omission and commission mismatches, as well a demonstrating that they are all eventually detected, the opportunity will be taken to show the process that might be followed to correct them.

The initial configuration is shown in Figure 6.17 and as expected there is a warning triangle on the component indicating that one or more mismatches have been detected.

As the stubborn connector type inhibits all message passing behaviour associated with a port no externally visible progress can be made by this component. Considering the choreography outlined in the previous section it is possible to deduce that this component would be initially willing to interact on ports s1, s3 and c3. Ports s1 and s3 are both inbound ports[7] and so it is expected that omission mismatches would be reported there. Port c3 is outbound and so a commission mismatch would be anticipated there.

[7]Inbound ports listen for the first message in their message pattern while outbound ports send the first message in their message patterns.

Following the choreography beyond the initially active ports reveals that the next ports in sequence after s1 are c1, s2 and then c2. C1 is outbound so a commission mismatch might be expected here, however the omission mismatch at port s1 means that the system will have no traces that reach c1, so the commission cannot occur at this point. S2 is inbound and so will be the locus of an omission. This event would normally be hidden by the analysis as it occurs after the commission on port c1, however as no trace can reach c1 there will be no deadlock trace and so the omission at s2 should be listed. No mismatches will be reported against port c2 at this point as the conversation thread cannot reach it due to the earlier deadlock at port s1.

Summarising the above, the following mismatches are expected to be reported:

Commission port c3

Omission ports s1, s2 and s3

Examining the rules view of the component reveals that both commission and omission mismatches have occurred, as expected, however the details of which ports exhibit those mismatches is a subset of those expected. The results from the analysis output follow.

Commission result

```
Broker attempted to send unexpected messages (commission events) in 1 traces.
Commission trace number 1
Broker_c3_sendReq
```

Omission Result

```
[Broker_s1_getReq]
```

These mismatches are consistent with those expected, they are also a correct assessment of the mismatches given the results returned by the FDR model. Once again we find that FDR is only reporting a subset of the dead and refinement failure traces that exist in the system.

The user of the system must now decide which of the reported mismatches to address. As we have already seen that omission results can be false, it is suggested that correcting commission mismatches first, and only when no more exist, should the user turn his attention to correcting the omission mismatches.

Following this principle, the commission mismatch related to port c3 should be tackled first. In this system there is only a single component attached to six stubborn connectors so the solution to mismatches at any of the ports is to change the connector type to a cooperative one.

Modification 1

The connector attached to port c3 is changed to a cooperative type and this leaves a system that now reports the existence of only a single mismatch as follows:

Omission Result

```
[Broker_s1_getReq]
```

This result confirms that the commission mismatch on port c3 has been corrected. As there are no other mismatches reported the architect should now move onto correcting the omission on port s1. Once again this involves changing the connector attached to that port to a cooperative type.

Modification 2

The third version of the system has two new mismatches reported.

Commission result

```
Broker attempted to send unexpected messages (commission events) in 1 traces.
Commission trace number 1
Broker_s1_getReq Broker_s1_sendFault Broker_c1_sendReq
```

Omission Result

```
[Broker_s3_getReq]
```

The first is a commission mismatch on port c1 while the second is an omission on port c3. Both of these are consistent with what would be expected. The commission mismatch was hidden by the earlier omission on port s1 that prevented the conversation trace reaching that port. The omission mismatch was also hidden by the mismatch on port s1 but, as already discussed, this was due to FDR returning only a subset of the expected traces.

Modification 3

Continuing with the commission before omission approach, the next modification made to the system was to change the connector attached to port c1 to a cooperative type. The resulting system also reporting two mismatches as follows:

Commission result

```
Broker attempted to send unexpected messages (commission events) in 1 traces.
Commission trace number 1
```

```
Broker_s1_getReq Broker_s1_sendFault Broker_c1_sendReq Broker_c1_getRes
```

Omission Result

```
[Broker_s3_getReq]
```

This is an interesting result as at first glance they appear to be the same ones that existed before the cooperative connector was attached in place of the stubborn one. This is certainly true of the omission mismatch which is identical to that reported before the change, however in the case of the commission mismatch there is now an additional message shown at the end of the trace.

The significant property of this additional message in the trace is that it is a message that particular component expects to receive, not one it expects to send. This means that the port c1 actually completed its message pattern successfully having sent and received a message but that these were the last two messages exchanged in the system.

This reveals a fault in the analysis logic as the port c1 is no longer harboring any mismatches however one is being reported against it. During the analysis each deadlock trace found is examined and if the final message in that trace is described in that component's interface then that component is considered to have sent it and therefore to have caused the commission. However this assumes that the only point where deadlocks can occur is after a message is placed onto the connector and before it is delivered to the other port. In this case port c1 sends and receives a message before the choreography moves to port s2, but the inbound port s2 is attached to a stubborn connector and so will never receive a message, meaning the system is deadlocked.

The fault in the logic stems from an over simplification used when determining if a commission event is caused by a particular component. Simply the analysis considers all messages in a component's interface when determining if that component sent the offending message, when in fact it should only consider the messages that component sends and not those it receives.

So the actual mismatch in the example is on port s2 but it is causing a false commission to be reported on port c1 and the false commission exists because of an assumption made during development of the analysis. In this case then the commission before omission principle breaks down as the commission result is a red herring and in fact the reported omission should be tackled.

Modification 4

After the connector attached to port s3 is replaced to address the above omission mismatch the system only reports a single mismatch as existing, this is the same false commission as discussed above.

Commission result

```
Broker attempted to send unexpected messages (commission events) in 1 traces.
```

```
Commission trace number 1
Broker_s1_getReq Broker_s1_sendFault Broker_c1_sendReq Broker_c1_getRes
```

There are no omission results reported by the analysis, even though it would be possible to demonstrate that one exists on port s2. The reason for this is that the traces leading to the omission contain the false commission trace and so the analysis is hiding it as a potential false negative.

At this point the modification required is not determined from the reported analysis but based upon a prediction of what the analysis would report if the commission assumption were corrected. Specifically this is that no commission mismatches would be reported while a single omission would be reported relating to port s2. This prediction is used to make the next change to the model.

Modification 5

With port s2 now connected to a cooperative connector, the result is a system in which a single commission mismatch is reported.

Commission result

```
Broker attempted to send unexpected messages (commission events) in 1 traces.
Commission trace number 1
Broker_s1_getReq Broker_s1_sendFault Broker_c1_sendReq Broker_c1_getRes
Broker_s2_getReq Broker_s2_sendFault Broker_c2_sendReq
```

This mismatch is one that would be expected to be reported by the analysis as it represents the message trace having finally reached port c2 and then being stopped by the stubborn connector it finds there. At this point then it is possible to return to following the commission before omission principle to correct the mismatch.

Modification 6

In the final iteration of this demonstration system the connector attached to port c2 is replaced with a cooperative type and no more mismatches are reported.

6.3.4.1 Section Summary

The main conclusions to draw from this section are that while the analysis works in most cases there are situations where an actual mismatch present can be masked. This results from all messages sent or received by a component being used to determine if it is the origin for an unexpected message when only the messages it sends should be considered. Time did not allow this flaw in the analysis code to be corrected within the scope of this work.

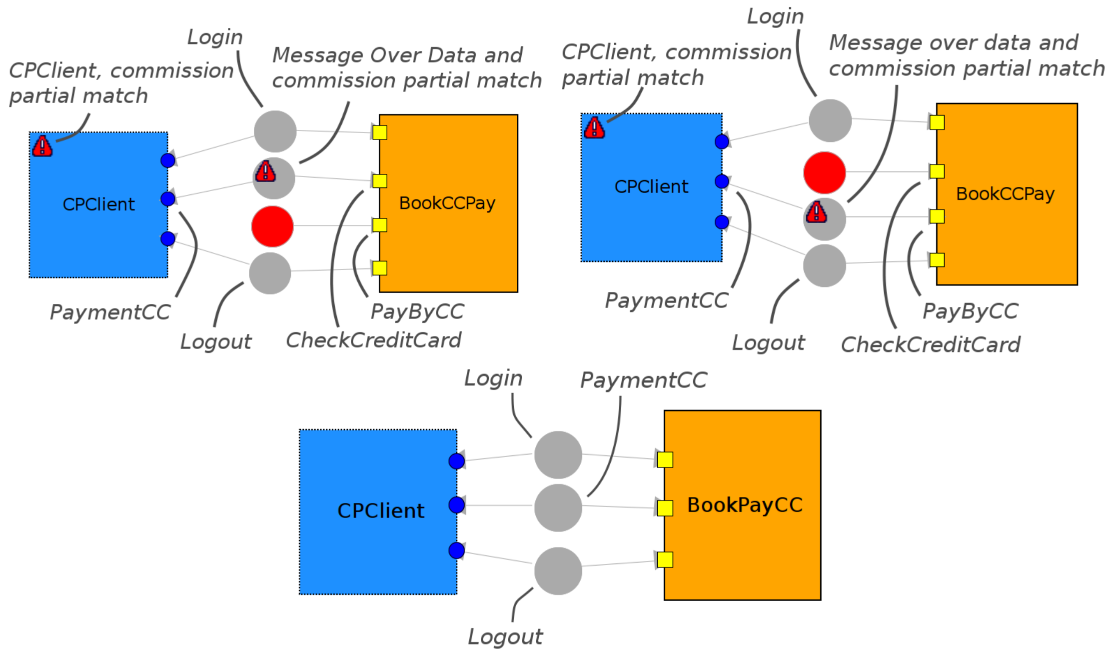


Figure 6.18: The three configurations used to confirm that the mismatched reported in the car parking scenario, Section 6.3.1, were not caused by the presence of multiple connectors being attached to individual ports.

If the above flaw in the analysis code had been corrected then the false commission indicated after modification 3 would not have occurred. Taking this into account then we argue that while FDR does not allow all mismatches to be detected in the first instance, if the correction to the analysis code were made and if the principle of "commission before omission" is followed then through repeated analysis and correction cycles a user will be guided to find all mismatches of those classes.

6.3.5 Additional Tests: Multiple Connectors

A demonstration that the style and analysis detects mismatches when multiple connectors are attached to ports has effectively been performed in the earlier car park scenario. However, it is important to know that the mismatches in that earlier model were genuine mismatches and not a side effect of the methods used to model the multiple connections. To demonstrate this, the models of the initial state of the car park scenario, in which mismatches exist, are dissected so that the car park client is connected to only a single service at a time. The three configurations of the client and both services are shown in Figure 6.18 and each of their ACME descriptions may be found in Appendix E.2.4. This shows that once again, there are no mismatches found between `CPClient` and the `BookPayCC` service but that there are mismatches found in both configurations involving the `SpaceCCBuy` service. Selecting the rule views for both these faulty configurations reveals that the following mismatches have been detected.

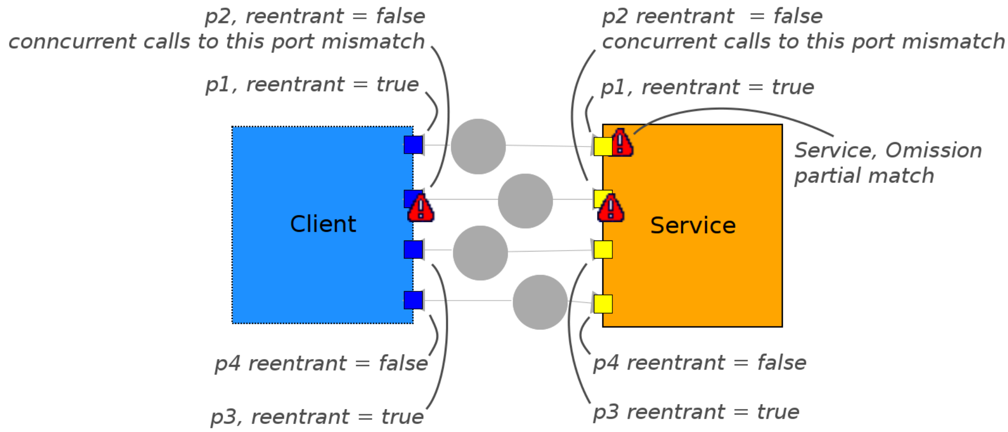


Figure 6.19: The system used to demonstrate the analysis looking for multiple threads in non-reentrant ports. The ports are named p1 ... p4 from top to bottom.

Initial configuration:

- Message Over Data - Message 1;
- Commission Partial Match;

Alternate configuration:

- Message Over Data - Message 1;
- Commission Partial Match;

These are identical to the mismatches found in the initial stages of the car park scenario. Furthermore, examination of the information in the analysis output reveals that details of the mismatches are also identical. This shows that the mismatches in the earlier car park scenario are not influenced by the presence of multiple connections.

6.3.6 Additional Tests: Multi Threading

To demonstrate the multithreading analysis included in the style a simple system consisting of two components was constructed, Figure 6.19, its ACME description may be found in Appendix E.2.5. Each component has four ports labelled p1 ... p4 and the central CSP in the components is set up so that p1 and p2 will experience multiple threads while p3 and p4 will only witness a single conversational thread. The expected conversations in terms of the port IDs are as follows.

$$Client \mathrel{\widehat{=}} Client\_Multi\_Thread \mid\mid\mid Client\_Multi\_Thread \mid\mid\mid Client\_Single\_Thread$$
$$Client\_Multi\_Thread \mathrel{\widehat{=}} p1 \rightarrow p2 \rightarrow Client\_Multi\_Thread$$
$$Client\_Single\_Thread \mathrel{\widehat{=}} p3 \rightarrow p4 \rightarrow Client\_Single\_Thread$$

$$\begin{aligned}Service \mathrel{\widehat{=}}\ & Service\_Upper\_Thread \mid\mid\mid Service\_Upper\_Thread \\ & \mid\mid\mid Service\_Lower\_Thread \mid\mid\mid Service\_Lower\_Thread\end{aligned}$$
$$Service\_Upper\_Thread \mathrel{\widehat{=}} p1 \rightarrow p2 \rightarrow Service\_Upper\_Thread$$
$$Service\_Lower\_Thread \mathrel{\widehat{=}} p3 \rightarrow p4 \rightarrow Service\_Lower\_Thread$$

As the client component contains all the ports that send the first message, this means that both the client and the service can have multiple conversations running through ports p1 and p2. At the same time, while the service could handle two conversations running through its lower thread (ports p3 and p4) this will not occur as the client has only a single thread to interact with those ports. To demonstrate that the analysis rules correctly account for the reentrance property of the ports, p1 and p3 on both components are defined as being reentrant while p2 and p4 are defined as not being reentrant.

The analysis returns three mismatches from this model:

- `Client.p2`: Concurrent calls to this port;
- `Service.p2`: Concurrent calls to this port;
- `Service`: Omission partial match;

The concurrent call mismatches are exactly as expected, firstly as p1 and p2 are the only ports that can actually experience multiple concurrent invocations and then the p2 ports are the only ones in that set that are not reentrant. The ports p3 and p4 are not shown as experiencing multiple threads as, while the service could service multiple invocations of those ports the client component only has a single thread working through those ports and so there can never be more than one actual invocation of each port at any time.

The third mismatch reported in this system is an omission of a message to port p3. This is consistent with the service being able to support multiple invocations, by way of it having two threads available to ports p3 and p4, but the client only ever utilises one of them.

6.3.6.1 Section Summary

This section has demonstrated that the enhanced style is able to represent a case study from the literature, confirm the problems stated about that system and show that the final proposed solution is devoid of mismatches. This showed that, with the notable exception of the false commission result, the analysis rules function as they were designed to. The subsection also showed examples of the analysis based upon CSP models and discussed some issues related to them.

As with the minimal style earlier, there is a need to consider what confidence can be placed in the style as a means for assessing web service systems in general. The approach taken with the minimal style, i.e. considering the scope of the analysis rules and the nature of the data upon which they depend, will also be applied in this case. The mismatches driving the enhanced style development were split into three groups, port to port scope, component to environment scope and type checking, the assessment here will follow the same groups.

Taking the port to port mismatches first, labelled cp1 – cp13 in Table 5.6. These mismatches are all constrained to consider the data contained in both the ports and components at either end of the connector. This means that this entire set of analysis rules is unaffected by the overall structure and size of the system being considered. The data for all of these analyses are based upon properties directly input by the architect into the connected ports and components, there is no manipulation of the data before analysis.

The second group of analyses included in the style are those associated with type checking, the analyses within this group have one of two distinct scopes. The first scope is constrained to within a component. There is only a single analysis with this scope and it is concerned with the types of ports a component owns. The second scope covers the whole system and these analyses consider the nature of components and connectors it contains. While these analyses have very different scopes they are joined by the nature of the data they act upon which are the boolean results of all analysis rules used to determine if an element satisfies its declared type or not.

The first two groups are both unaffected by the quantity and structure of components and connectors in a system under analysis and so the suggestion is that these analyses would perform equally well on any web service system described according to the style.

The final group of mismatches are those with a component to environment scope. All four of the analyses produced to address mismatches in this group are based upon the generation of CSP models from the CSP fragments included in the components. These models are then checked against a specification, such as deadlock freedom or being a correct traces refinement of some model, by the FDR model checker. This means that they are in some ways the complete opposite of the previous group as each analysis required data from the entire system and also that data is manipulated by the analysis code before being passed to the model checker.

This last group of analyses is affected by the structure of the system as the data they act upon is generated from properties within each component and so there are two potential points at which they could fail to perform correctly, the generated models upon which each analysis is based could be incorrect in some way and the analysis performed upon the model could be fundamentally flawed. Taking the model generation point first, while the analysis code was tested using a number of test systems during the development of the analyses and these systems targeted specific aspects of each analysis, this testing is in no way guaranteed to be complete in terms of all combinations of port types, multiple connected ports, central CSP templates, cooperative and stubborn connectors etc. This means there may be system configurations that result in a model being produced that is not correct with respect to the system being analysed and the analysis to be performed. Secondly is the nature of the model checking and interpretation of their results. Each one was discussed individually in Chapter 5 and they stand on those arguments alone. They were also tested during development, but as with the model generation and as highlighted by the false commission result in Section 6.3.4, these tests cannot be guaranteed to cover all situations. The result is that there must currently be a degree of doubt placed upon the results output by this group of rules, but this doubt can be reduced by correction of the known faults.

6.3.7 Mismatch Coverage by Examples

In this section, the detection of a number of mismatches has been demonstrated and discussed. There were two motivations behind the tests selected for inclusion in the work. The first was to explore the effectiveness of the CSP based mismatch detection as the modelling of the message passing behaviour was the most complex part of the style and would be interesting to demonstrate. The second motivation was to demonstrate that the style could be used to support the related work of Cavallaro and Di Nitto [CN08].

The examples shown achieved those goals but they do not cover all mismatches identified in the early part of Chapter 5. Table 6.4 returns to the list of mismatches and shows in which sections the individual mismatches have been demonstrate. From this list we can see that the majority of mismatch types are listed as “tested during development”, meaning they have not been explicitly tested in this thesis. Each and every one of those mismatches listed were tested during the development of the style and, with the exception of a few that are discussed later in this subsection, they all worked as expected. The motivation behind not including the systems in which these were tested was brevity. The excluded systems tested mismatches that were detected using relatively simple techniques such as a set comparison, so it was considered that their inclusion would add little to the value of the work while adding considerably to the bulk.

In each case the excluded mismatch analyses were checked by constructing trivial systems that allowed a range of values for the properties of interest to be tested. The actual range of values used to check each mismatch analysis depended on the nature of the analysis. An example is the case of the failure modes mismatch (Section 5.2.1.7), where the analysis is essentially a comparison of two sets. In this case only a small number of the possible values for each set concerned were tested, just enough to give confidence that the ACME Studio “isSubset” function worked correctly. Other mismatch analyses were tested with a complete set of inputs. An example of this is the connector

ID	*Description*	*Section Demonstrated*
	Port to port scope	
cp1	Mismatching message exchange patterns	Mimimal style version demonstrated in Section 6.2, Enhanced style version tested during development
cp2	Partially matching message exchange patterns	Mimimal style version demonstrated in Section... Enhanced style version tested during development
cp3	Incorrect binding time of a service provider	Tested during development
cp4	Differing data continuity assumptions	Tested during development
cp5	Mismatching data types in a message	Tested during development
cp6	Mismatching data structure/syntax	6.3.1
cp7	Mismatching data semantics in a message	6.3.1
cp8	Mismatching state maintenance assumptions	Not addressed
cp9	Mismatching state scope assumptions	Tested during development
cp10	Mismatching failure mode assumptions	Tested during development
cp11	Mismatching connector creation/destruction assumptions	Tested during development
cp12	Connection to a non public web service port	Tested during development
cp13	Connected ports must share transport and encoding protocols	6.2
	Component to environment scope	
cc1	Concurrent calls to a no queuing and non-reentrant port	6.3.6
cc2	Mismatching conversations	6.3.1 & 6.3.2 & 6.3.3 & 6.3.4 & 6.3.5
cc3	Partially matching conversations	6.3.1 & 6.3.2 & 6.3.3 & 6.3.4 & 6.3.5
cc4	No component has an active thread of control	Not addressed
cc5	Concurrent threads in a single thread only component	Not addressed
cc6	Concurrent threads in a non-reentrant port	6.3.6
cc7	Mismatching process distribution assumpations	Not addressed
	Type checking	
ct1	Non web service compliant connector	Tested during development
ct2	Non web service compliant component	Tested during development
ct3	Ports must be well defined	Tested during development
ct4	Components must have correct port types	Tested during development
ct5	Components must be well defined	Tested during development
ct6	Connectors must be well defined	Tested during development
ct7	Roles must be well defined	Roles have no properties so no well defined checks are performed

Table 6.4: The sections in which the detection of specific mismatch classes by the enhanced style is demonstrated.

creation and destruction assumptions (Section 5.2.1.8), here it was necessary to test all combinations of the property values to gain confidence that the logic the rule was based upon returned the expected results.

A third type of analysis to test is exemplified by the mismatching data semantics in a message rule (Section 5.2.1.2). In this case the analysis rule takes into account the semantics of each datum included in each message exchanged. There are no fixed values defined by the style for these semantics, the intention being that these values would be defined in one or more ontologies, this meant that it is not possible to test all possible values. The testing in this case took advantage of the fact that the actual values of the semantics are not important, they are treated simply as strings, but whether the value declared by one component and the value expected by the other are equal is. So as the semantics are represented as strings the standard Java sting comparison methods were employed to check for equality. This meant that the required testing was reduced to a small number of cases where the quantity of data items and the simple strings representing their semantics were varied to confirm the logic of the external analysis code functioned correctly.

Earlier in this section it was mentioned that there were a few parts of the analysis that did not work as expected, specifically these were the two global rules concerning the architectural element types that exist in the system and rules confirming that a string property is populated. The two global rules make use of the ACME function "satisfiesType". In ACME Studio version 2.2.9b these rules worked correctly, i.e. a component would only satisfy it's type if it contained all the properties required and all rules relating to that type are passed. The rules were carried forward into the enhanced style which made use of a later version of ACME Studio, 3.2, and its support for external analysis. Unfortunately in ACME Studio 3.2 the satisfiesType function does not account for the results of the style rules defined for each element type. The result is that the output of the global rules checking element types in a system cannot be trusted in version 3.2, but it is hoped that this software bug is corrected in future versions of ACME Studio.

The second analysis part that did not function as expected is the check that a string property is populated. Once again this is a carry over from the minimal style in ACME 2.2.9b, in which is does work correctly, resulting in a rule being failed if a string is empty or is not defined at all. In ACME Studio 3.2, the rule results in an error if the property in question is not defined. In a sense this still has the effect of alerting the architect to the fact that a required property is not defined, but does not have the feel of being a proper check.

6.4 Evaluating Mismatch Detection in the Enhanced Style

The previous section demonstrated the enhanced style analysis using a number of test systems to show how well it performed in the task of detecting mismatches. In this section, a number of different views of the style and its analysis will be presented. These views challenge the depth of the analysis, the correctness and meaningfulness of the results it returns and how dependant the analyses are on having a complete model.

6.4.1 Depth

In this analysis depth refers to an estimation of how much the analysis tells the architect that he does not know by just looking at the model. For example an analysis rule confirming that a boolean property has been populated can certainly help ensure that a model is correctly described but does not tell us anything that would not be known by looking for that property in the model. This could then be described as being a shallow analysis. At the other end of the scale, a CSP model created from a number of fragments and used to determine if all traces of a component are allowed by the emergent behaviour of the system it exists in could be described as being deep.

The actual values of shallow and deep are certainly subjective and so no attempt to assign numerical values is made. Instead, the analysis rules are presented in groups where all the rules have an arguably similar depth.

As well as being separated by depth, the groups of rules are also separated into those that provide analysis directly relating to a mismatch type and those that confirm the model produced has the correct element types with all properties described and within the prescribed value ranges.

These groups are now presented, starting with the mismatch oriented rules and then the style oriented rules.

Mismatch Oriented Rules

For the mismatch oriented rules four groups of rules have been identified and will now be presented in order of increasing depth.

The first and shallowest group of mismatch rules are those that simply compare the values of two properties to determine if a mismatch exists. These constitute seven of the 33 mismatch rules in the style as follows:

- there are four rules confirming that each of the allowed port types has an acceptable number of connectors attached to it;
- two rules confirm that a connector is attached to one outbound and to one inbound port;
- the final rule confirms that the data continuity assumptions of a pair of connected ports match.

The second group of rules are those that require comparison of multiple properties to determine if there is a mismatch. Three of the 33 mismatch rules fit into this category:

- two of the rules are used to compare the message exchange patterns employed by a pair of connected ports, if a mismatch is found it requires the `inOurControlDomain` property to be looked up to classify it as being a partial match or a mismatch;

- a single rule compares the values of the connector creation and destruction assumptions of a pair of connected ports for compatibility.

The penultimate group of mismatch oriented rules is also based upon comparing multiple properties, but now the properties contain a data structure rather than single values. This means searching through the structures to find the required data instances. Also included here is a rule that requires comparison of two sets to determine the presence of mismatches.

These are described as being deeper than the previous group of rules because there is an increased overhead in searching for the values to compare before comparison can be made. This represents an increased opportunity for mistakes to be made if the analysis were performed manually rather than using a supporting style such as this.

Eighteen of the 33 rules in the style are assigned to this category:

- there are four rules in the common connector type that compare the data types of the data included in the messages exchanged. This requires gathering and matching message names from the message pattern properties of the connected ports. The names are matched to the messages in the messages property so the individual data they include can be matched based upon their semantics in the central data store. Once matched the data types of each datum can be looked up in the messages property and compared. There are four rules performing this, one for each possible message in a message exchange pattern:

- similar to the data types above, there are four rules comparing the semantics of each datum in each message exchanged by a pair of ports. The semantics are compared in the “over data” rules to check for redundant data being sent;

- there are a further eight “under data” rules comparing the semantics of the data in the messages exchanged. In this case the rules check for omission of data expected by the port receiving the message;

- also linked to the data exchanged is the “state scope assumptions” rule. Here each matched datum in the exchanged messages has its expected and exhibited state scope assumptions looked up and compared to determine the existence of mismatch;

- the final rule in this group compares the sets of expected and exhibited failure modes declared by a pair of connected ports, to pass the rule both exhibited sets must be subsets of the other port’s expected sets.

The final group in this category is unique among the analysis in the style in that the analysis results they provide cannot be determined directly by observing the values of properties. In this case the values of a number of properties along with the very structure of the system itself is used to construct a model that is then checked against specific assertions to determine if certain types of mismatches exist. For this reason this group is considered to be the deepest of all the analyses. Five of the 33 rules are found in this category:

- two of the rules generate models checking for commission mismatches relating to a specific component;
- two of the rules generate models checking for omission mismatches relating to a specific component;
- the final rule generates a model checking to determine if a specific port experiences concurrent invocations or not.

Style Oriented Rules

As with the mismatch oriented rules, four distinct groups are identified in this category. While it could be argued that the first three groups presented follow a pattern of increasing depth, the ordering of the last two is less distinct. The separation between them is, as we will see, based upon the sort of analysis they perform rather than the complexity of the analysis.

The first group in this set of four contains those rules that confirm a property has been populated. These constitute 15 of the 25 style oriented rules[8] in the style. These are listed below but no description is given as the names give an adequate indication of the property they target.

Components `CentralProcessDescribed`, `ComponentInOurControlDomainDescribed`;

Ports `EndpointListPopulated`, `InOurControlDomainPopulated`, `ReentrantPopulated`, `BindingSelfAddPopulated`, `BindingSelfRemovePopulated`, `BindingOtherAddPopulated`, `BindingOtherRemovePopulated`, `MessagePatternPopulated`, `DataContinuityPopulated`, `BindingTimePopulated`, `ChoiceGroupPopulated`, `EndPointAddressPopulated`, `HasWSDL`.

The next group of style oriented rules are those that compare multiple parameters to determine if the model is correct to the style. Only two of the 25 rules are positioned here.

- a single rule confirms that all service end points are addressed by asserting that the cardinality of end point definitions and end point addresses are equal;

[8] The total number of rules is based upon the properties they consider, not the number of rule instances in the style. For example, each port type has a rule to check it has some ports but the rule performing this was only counted once.

- the second rule confirms that the connector creation and destruction properties in each port are populated in a sane manner so that at least one port may create the connector and also at least one port may destroy it.

The third group of rules are those that require the consideration of multiple elements or the exploration of multiple properties. These all required the use of the ACME Studio external analysis feature for their development. There are three rules in this category.

- The `MsgDatumDescribed` rule confirms that each data item in each message of each port has an associated entry in the central data store to allow its semantics and scoping data to be extracted;

- a single rule confirms that each set of ports on a component that share a choice group have at least one port designated as the choice maker;

- the final rule in this set confirms that the message names included in a port's message pattern CSP description also exist in that port's messages property.

The final group of rules perform type checking on the elements in the style. While the structure of the rules in the style are relatively simple these rules utilise the Armani predicate *satisfiesType(X)*. For this predicate to return true, the element being checked must pass each rule associated with type X and also have every property required by type X. There are five of the 25 rules involved in this type of analysis.

Component `ComponentHasValidPorts`, `AllClientPorts`, `AllServicePorts`, `ComponentHasClientInterface`, `ComponentHasServiceInterface`[9];

Connector `CorrectNumberOfRoles`;

System `NatureOfComponents`, `NatureOfConnectors`.

6.4.1.1 Section Summary

While many of the rules in the enhanced style involve analysis that could be performed by manual inspection of the component and port descriptions it should be noted that the style leads to models containing a great many properties, for example, the final configuration of the car parking scenario contains 300 property instances. Given this, it is suggested that there is a distinct possibility that a manual approach would result in some mistakes and so argue that the rules add value.

[9] The `AllClientPorts`, `AllServicePorts`,`ComponentHasClientInterface`, `ComponentHasServiceInterface` rules all perform a similar function but are specialised for the component types. They are counted as a single rule for the purpose of this analysis.

The model checking based analysis, while in the minority of the rules, also adds to the value of the results by informing the user of mismatches that can only be determined by discovering the emergent behaviour of the complete system.

6.4.2 Dependancies

It is generally acknowledged that the earlier in the software development process that a fault is discovered the cheaper it is to correct [PA06]. This principle, we suggest, could also be applied to the development of a system architecture model representing a future system, in this case though, the faults take the form of architecture mismatches. If this idea is accepted then it follows that it is desirable if mismatches can be detected at the earliest possible point when developing the architecture, i.e. as soon as the properties and structure required to determine each individual type of mismatch are in place.

Assessing how complete the architecture model needs to be before each analysis rule in the style can be evaluated results in three distinct groups.

The very best rules, those that can be evaluated earliest, are those that focus upon whether the properties required by the style exist and are populated. These are only dependant upon a single or just a few properties within the same port and component. They are all able to confirm if a component has been correctly populated before any connections are introduced in the system.

Component `CentralProcessDescribed`, `ComponentInOurControlDomainDescribed`, `ComponentHasValidPorts`, `AllClientPorts`, `AllServicePorts`, `ComponentHasClientInterface`, `ComponentHasServiceInterface`;

Port `EndpointListPopulated`, `InOurControlDomainPopulated`, `SendsFirstMessagePopulated`, `ReentrantPopulated`, `BindingSelfAddPopulated`, `BindingSelfRemovePopulated`, `BindingOtherAddPopulated`, `BindingOtherRemovePopulated`, `MessagePatternPopulated`, `DataContinuityPopulated`, `BindingTimePopulated`, `ChoiceGroupPopulated`, `EndPointAddressPopulated`, `EachEndPointProtocolAddressed`, `HasWSDL`, `StatedBindingTime`.

The next group covers the majority of the analysis rules. This includes the remaining analysis rules that are purely based upon the Armani predicate language and do not make use of the external analysis. It also includes the external analysis based rules that consider the messages exchanged between ports in terms of the data types and semantics they contain. All of these analyses can be performed as soon as their required properties are populated and there is a connector relating the ports to be assessed.

Port `EndPointProtocols`, `OnePortSendsFirstMessage`, `OnePortReceivesFirstMessage`, `MatchingDataContinuityAssumptions`, `MsgXMessageDataTypesMatch`[10], `MsgXMessageOverData`, `MsgXMessageUnderData1`, `MsgXMessageUnderData2`, `ConnectorCreationDestruction`, `SaneConnectorCreationDestruction`, `FailureModeAssumptions`;

The final group includes all the remaining external analysis rules. These can only be performed when the whole model is complete and populated with data. This means a mismatch revealed by any of them may incur the maximum cost to repair in terms of correcting the architecture.

The reason for this dependancy on a complete model for evaluation was a decision made while constructing the external analysis. The decision was to build a simplified Java version of the architecture to reduce the programming overhead involved in extracting each property from the ACME model when needed. This made developing the external analysis easier, however the Java model requires that many of the properties of the system are populated before it can be constructed. This means the analysis that uses it is also dependant on these properties being populated before they can be evaluated, even if a particular analysis does not require some of the properties for its evaluation.

The result then is that this last set of external analyses are sometimes artificially delayed by waiting for properties to be populated when they don't need them to be.

Component `CommissionMismatch`, `CommissionPartialMatch`, `OmissionMismatch`, `OmissionPartialMatch`, `MssgDatumDescribed`, `ChoiceGroupsHaveChoiceMakers`

Port `PortReentered`, `MsgNamesConsistent`

Connector `MessageExchangePatternsMatch`, `MessageExchangePatternsPartiallyMatch`, `StateScopeAssumptionsMatch`

6.4.3 False Results

During the development and analysis of the enhanced style, a small number of possibilities for both false positive and false negative results have been identified. These will now be presented.

6.4.3.1 Hidden Commission

As discussed during the earlier demonstration of the omission mismatch analysis, Section 6.3.2, the FDR model checker does not necessarily return all traces leading to deadlock when it is assessing a system for deadlock freedom. The result of this is that while the analysis will always report the presence of a commission event if one or more exist in the system, it may not report them all in

[10] Here the X refers to the message number in the message exchange pattern. There are up to four messages described in a pattern and so there are four copies of each of the rules starting `MsgX...`.

the first instance. So the result of altering a model and removing a commission event could be the revelation of further commission events that already existed in the system.

6.4.3.2 False Commission/Hidden Omission

This false result is shown in the stubborn connector demonstration in Section 6.3.4. This situation occurs because there is a solicit-response port, c1, attached to a cooperative connector and this is followed by a request-response port, s2, attached to a stubborn connector. This means that s2 will never received the incoming message it is waiting for, in terms of the analysis, this constitutes an omission event. However the omission analysis aims not to report potentially false negative omission events by hiding those that occur after a deadlock. In this system there is a deadlock immediately after c1 receives a response message from the cooperative connector as s2 cannot proceed, resulting in the omission on s2 being hidden. The problem here is that the analysis assumes that deadlocks can only occur when a port sends an unexpected message i.e. a commission event and that the last message in a deadlock trace will be the unexpectedly sent message. This can only be resolved by altering the analysis to filter deadlock traces where the final event is a message received by a component and not one sent by a component.

6.4.3.3 Hidden Omission

In this case the hidden omissions are caused by FDR not returning all the refinement failure traces it can find. As with the hidden commission earlier, the analysis will report the presence of an omission event if one or more exist, but it may not report them all in the first instance. Again this means that the removal of an omission reported by the analysis may result in a further example being reported.

6.4.3.4 Potentially False Omission

As omission mismatches are detected via a refinement assertion it is possible for the FDR model checker to find an omission that occurs after the system being analysed has deadlocked. It is termed a ‘potentially’ false omission due to it occurring after the system deadlocked and therefore the omission may be a genuine mismatch or it may be a result of the prior deadlock. To protect the architect against being inundated with potentially false omission results, each omission mismatch found is checked to see if it occurs after a deadlock, if it does this omission is not reported. The problem observed in Section 6.3.2 was that while the omission did occur after deadlock, FDR did not return all traces to the deadlock or the following omission so the this was not detected by the analysis.

6.4.3.5 Omission Partial Match/Mismatch

To detect an omission mismatch, an assertion in the CSP model of the system that the system is refined by the component in focus is used. For this to work it is necessary to hide every message in the system that is not defined in the interface of that component. The result of this is that, while the refinement trace can inform the analysis of which message was omitted, it does not inform the analysis of which component and port was expected to send that message. The result of this is that the analysis is not aware of whether the sending component and port are `inOurControlDomain` or not.

The analysis currently assumes that for the receiving port `inOurControlDomain` has the value `No`, then this means that the analysis has to report an omission mismatch exists when it could in fact be a partial match if the other component is `inOurControlDomain`. This could be corrected by altering the analysis code to look at the port on the other end of the connector, however this approach can only work where a single connector is involved, ports with multiple connectors attached would still exhibit the same problem.

6.4.3.6 String Properties Correctly Populated

There are no facilities in ACME Studio to parse a string property, thus the analysis rules charged with confirming a property is populated can do only that. They are able to report if there is a string value in the property or not, they cannot confirm if that string is meaningful. There are varying degrees of need for this function, for example the choice group names in unicast ports can use a short string for the name, however the properties that hold CSP could benefit in terms of checking the CSP is syntactically correct and the process names match across the whole component. This functionality could be added in the form of further external analysis classes.

6.4.3.7 Global Type Checking Rules

There are a number of type checking rules in the style that make use of the ACME Studio function “satisfiesType”. In ACME Studio version 3.2, used for the developement and testing of the enhanced style, this function is flawed and returns the result “true” regardless of the outcome of any rules relating to the element type in question. This means that an element may declare itself as a web service connector type and therefore be bound by all the rules defined for that type, but it may fail to meet the conditions of any of the rules without the global rule checking that all connectors satisfy the web service type reporting a fault. It should be noted that while this is not ideal, the architect will still be alerted to the fact that the connector failed the conditions of one or more rules by the red warning triangle that will appear on the connector itself.

The nature of this false result means that it is out of this author’s control to correct, but it has been brought to the attention of the ACME Studio developers and it is hoped it will be corrected in future versions.

6.4.3.8 Discussion

The issues listed above are not equal in terms of the size of problem they pose to an architect using the ACME Studio with the style. As the purpose of the style is to facilitate mismatch detection it follows that we may rank these issues in terms of their effect on the accuracy of the mismatch detection. The two false results with the least significance on this scale are the hidden omission and hidden commission. These are the least significant issues as the architect may accept that they exist and know that correcting the current set of reported mismatches may result in previously hidden instances being revealed. If this process is repeated then eventually all instances of these mismatches will be revealed and can then be corrected by the architect. The potentially false omission issue is also at this level of significance as it can be worked around by removing all commission mismatches before addressing any omission mismatches.

The omission partial match/mismatch issue would appear next in the significance order. For the architect to determine if the reported mismatch is in fact a partial match requires the examination of the 'in our control domain' property of the port that should have sent the message. So if the port expecting the message only has a single connector attached then this can be done with ease. The difficulty arises if there are multiple connectors attached to the port, in which case it may not be possible to identify which other component was being interacted with and should have sent the message.

The most significant of the issues is the false commission/hidden omission false result. This has been placed at the top of the list as there is no simple approach to addressing the issue as it requires an understanding of the CSP descriptions of a component and the system to diagnose its existence. Fortunately this issue can be removed from the style by an adjustment of the analysis code as described in Section 6.3.4.

The final issue listed above is given a separate treatment as the problem it miss-reports is the failure to populate a property rather than a mismatch that would appear at run-time. The problem associated with it failing, i.e. it reports that the string property is populated but does not report some fault in the value of that property, is that there is no systematic process for discovering the nature or location of the fault. For this reason it is considered to be a potentially significant false result due to the time that may be lost in diagnosing it. As stated above, this issue could be removed by adding further analysis classes to parse the string properties and check for consistency with any other related properties, such as port names being typed correctly.

The above list was populated from notes made during testing and evaluation of the style and so is complete in that sense. There may of course be other issues that have not yet been discovered but it is hoped that the testing revealed the most prevalent items.

6.4.4 Meaningful Results

Another view that can be applied when assessing the enhanced style regards how meaningful are the results of the analysis. The results presented within ACME Studio itself take the form of a three value boolean that equates to passed, failed or indeterminate, where this latter value indicates there was some problem evaluating the rule.

The rules within the style can be separated into two sets based upon their implementation method, those that are implemented entirely using the built in Armani predicate language and those that use the external analysis facility to some extent. These should be separated as the opportunities for providing a meaningful result differ vastly between the two. The first group examined are those that only use the Armani predicate language and so can only respond with a boolean value, then the rules using the external analysis and therefore able to output text files with detailed results will be examined.

In both cases the rules will be classified as being “OK” or “Expansion Required”. The second grouping implying that, perhaps the meaning of the mismatch in terms of the course of action that could follow is not clear. In this latter case, some description of why the description falls short is provided.

6.4.4.1 Armani Only Rules

A great number of the Armani rules that were considered to be OK were concerned with checking that the properties were populated, but this set also included some of the mismatch detection rules. However, the boolean nature of the Armani rule output combined with some limitations of the language mean that several of the rules fall into the expansion required classification, these will be detailed now.

Component Rule: Central Process Described This rule checks that the string property is populated, however this property contains one of the CSP descriptions of the system and so even if populated the contents could be both syntactically incorrect CSP and also inconsistent with the other CSP properties in the component.

Port Rule: Message pattern populated This rule checks that the string type property, message pattern, is populated. This contains the other parts of the CSP model mentioned above and it potentially suffers from the same problems, that a positive result can be achieved while it contains an incorrect and inconsistent CSP description of that part of the component.

Connector Rule: Matching end points The purpose of this rule is to confirm that two connected ports share a pair of protocols to encode and transport the messages they exchange.

The rule returns a positive result if there is one or more shared pairs of protocols, however this does not indicate which pairs of protocols match. This mandates a manual examination of the related data structures in both ports to determine which protocol pairs can be used.

Port Rule: End points addressed A second rule relating to the end points aims to ensure that there is an address defined for each end point. The rule simply counts the number of end points and addresses and returns a positive result if they match. A fail result then is caused by a difference in the cardinality of the end point addresses and end point protocol sets, while this could also be the result of multiple end point addresses employing the same protocol pair. Connecting the end point protocol and point address data structures would facilitate a more meaningful result if external analysis were employed and would also provide extra data to improve the results of the previous rule.

Connector Rule: Failure mode assumptions This rule compares the expected and exhibited failure modes of a connected pair of ports. A fail result is returned if either of the exhibited sets is not a subset of the opposing expected set. The expansion opportunity would be to indicate which failure modes are missing from the expected set rather than expecting a manual examination of the sets as the current rule does.

Port Rule: Has WSDL This is another rule checking that a string property is populated. Again, if parsing were possible, then the rule could differentiate between the property being devoid of a value and it being an incorrect url. Another possibility is that a future implementation of this style could include a rule that verifies the stated url of the WSDL document by fetching it.

Connector Rule: Connector creation/destruction assumptions This rule gives a negative result if any of the four connector creation/destruction assumption properties do not match. The improvement here would be to indicate which properties formed the basis for the mismatch and what their values were, again this is to remove the need for manual inspection of the model following the analysis.

Component Rule: Component has the right type of ports This rule confirms that a component is populated with ports that satisfy the types it should have, the rule returning a negative result if any of the ports are incorrect. At first glance this appears to be another instance of a rule failing that requires the inspection of, in this case, the declared port type. However this is not necessarily so as the rule can be passed so long as the ports hosted by a component have the correct properties and pass the rules of the required types, they do not have to declare the type. Thus, to reduce the effort involved in searching, it would be advantageous if this rule reported back which port types were missing or which specific ports failed to meet the requirements.

System Rule: Contains the correct component types This rule is similar to the above, however in this case the rule should return the identities of the offending components.

System Rule: Contains the correct connector types As above, but with the identities of the offending connectors listed.

6.4.4.2 Armani and External Analysis Rules

The external analysis method of performing analysis involves the creation of Java plug-ins compatible with the Eclipse environment on which ACME Studio is built. So while these analyses are limited to returning a boolean response to the user within the ACME Studio environment, they can also output further detailed explanations, in this case, through plain text files. As such, when considering the output of each analysis, it was found that the vast majority of them, 13 out of 15 classes, did produce output that could be used to direct corrective actions. A description of all the analysis outputs can be found in Appendix F.2 starting on page 310.

The two analysis classes that return an output that would benefit from some expansion are both involved in the omission analysis, one detecting mismatches and the other reporting on partial matches. The output takes the form of the trace followed by the component in focus up to and including the omitted message. While this informs the user about the behaviour of the component in focus it does not give any detail about the behaviour of the surrounding system other than that it is unwilling to send the missing message at this point.

In a system consisting of two components both with single conversational threads, then it would be possible to determine the behaviour of the other component based upon the trace information. However this task increases in difficulty as the number of components and conversation threads increases due to the number of traces that need to be explored. Some tool support to assisting with the exploration of system traces leading to the refinement failure point could greatly assist with understanding the state of the other components in the system and from that potentially lead to solutions to the mismatch being derived.

6.4.5 Scope of the Enhanced Style

In Section 2.2.3.3, page 30, a definition of architectural specification by Eden and Kazman [EK03] was presented where architectural characteristics were said to be *intensional* and *non-local*. An aide-mémoire to the definitions and their example is given below:

Intentional specification a specification is intentional if it can be satisfied by an unbounded number of programs.

Non local specification a non- local specification is one that it can be satisfied in "some corner" of a program without being affected by what the rest of the program is like.

The example Eden and Kazman gave was a layered architectural style with two rules.

- each element must be described in exactly one layer
- each element may only depend on elements in the same layer or lower layers

This style meets their definition of architectural specification as there are an unbounded number of programs that may meet the specification (intentional) and any one component failing to meet the second rule means the system as a whole is not correctly characterized by the style (non-local).

The focus of this part of the evaluation is to discuss whether the mismatches presented in this thesis are also intentional and non-local in nature and so can justifiably be termed architectural mismatches.

Non-local

The above layered style could be altered to include a "layered element" component type, then this component type could include the two rules regarding where an element is described and which components it depends upon. The style would then need a rule stating that all elements need to satisfy this type for the system to be considered correctly layered. The modified style is shown below:

- layered element rule - this element must only be described in exactly one layer
- layered element rule - this element may only depend on elements in the same layer or lower layers
- all elements in the system must satisfy the type layered element

This style is identical in effect to the original one in that any one component failing to meet either condition will result in the system not meeting the requirements of the layered style specification. In this respect, the web service architectural styles presented in this work are identical to the second style. Both web service styles define component and connector types containing rules that must be respected for the architectural elements in the system to satisfy their types, also both styles contain global type checking rules requiring that all elements in the system respect the component or connector types defined. So if a single component or connector fails to meet the requirements of the element type then one of the global type checking rules will not be passed and the system as a whole will not be considered to meet the requirements of the web service style. Therefore, even though many of the rules in the style have a local scope, some even just considering properties within a single port, the overall style can be considered to be non-local in nature.

Intentional

The layered style example was said to be intentional as it could represent an unbounded number of programs Eden and Kazman stated that this was obvious [EK03], but it is presumably due to there only being a single constraint on the elements run-time properties, specifically that it only depends on elements in layers below it. The layered style therefore imposes no constraints on the variables representing the inputs to the elements, their internal processes or their outputs, so these could be said to be free.

The enhanced architectural style does include properties and rules relating to both the inputs and outputs of each component in terms of their data types, semantics and choreography but it does not enforce any bounds on these, for example there are no bounds on the number or semantics of the data items input or output by any port. There are bounds on the pattern of messages that may be exchanged by any one port, but since there is no upper limit to the number of ports a component may possess and those ports may be linked together to form longer patterns of message exchanges it follows that there are no bounds on the number of program models a component in the style could represent. Furthermore, the style places no limits on the number of components that may exist in the system.

At the same time, there are a number of characteristics that are tightly constrained, such as the message transport and encoding protocols a port may employ, these are limited to a number of versions of HTTP and SOAP respectively. It would be fair to say that the choice of using, for example HTTP 1.0 or HTTP 1.1 as the transport protocol, is an implementation level specification as it is potentially local to an individual port and constrains the port to some degree. The same argument could also be applied to the port property describing the choice of SOAP versions supported by a port.

The argument then is that while the components are constrained in certain aspects of how they communicate they are not constrained in terms of what they communicate or how they process that data and so can indeed represent an unbounded number of program models. Based upon this argument it is fair to say that the enhanced architectural style presented in this work is justified in using the architectural term with respect to the majority of the analyses performed, while at the same time it is arguable that the implementation level characteristics could be removed from the style to leave a model that is more purely architectural.

6.5 Summary

The minimal style showed that it is capable of representing a system and detecting the classes of mismatch within its scope and so in some sense it meets its requirements. However, as discussed in Chapter 5 the data structures employed were poor as they required data replication which introduces the possibility of inconsistency.

This chapter then showed that the enhanced style is also capable of representing a system and detecting mismatches within it. In this case the system was drawn from the literature and it demonstrated the style's ability to support related work by detecting existing mismatches and confirming that they were removed in the final configuration.

The additional smaller tests raised some key points about the analysis with respect to implementation assumptions and the external model checker employed. Both of these issues can be either corrected by modifying the source code of the analysis plug-ins or by employing the "commission before omission" principle.

Assessing the style from a depth viewpoint informed us that while the majority of the analyses included in the style could be performed manually, there are two distinct benefits of employing the style. Firstly, the task of performing the commission, omission and multi-threading analyses is not practical for a non trivial system without the aid of tool support. Secondly, the number of individual properties and mismatches to consider would make mistakes a distinct possibility.

In terms of dependencies it was seen that a large number of the analyses were being artificially delayed by an implementation decision, potentially increasing repair costs if mismatch is found. This again could be corrected by adjusting the analysis source code.

The results were found to be mainly meaningful, especially when the external analysis was employed. This raises the question about whether the decision to employ external analysis should only be based upon complexity of the analysis and whether a reasonable Armani predicate can be formed, as was the case when developing the style or whether the detail desired from the results should also play a part.

Finally, with respect to false results: some can be adjusted by correcting the analysis source code, while others that stem from the FDR model checking output can be mitigated by following a procedure.

With the styles assessed, it is now possible to consider what future work exists in this area, the details of which will be discussed in the following chapter.

Chapter 7

Further Work

The future work below is divided into two sections, the first discusses details of future directions for the enhanced web service architectural style, while the second section touches on work relating to the SOA aspects of the thesis.

To guide the reader potentially interested in performing any of the future work. the description of each item is followed by a brief discussion of both its value to the work and the type of effort that is believed to be involved. The value is divided into one of two categories. *Substantive* modifications are those that would yield improvements in the analysis performed by the style, while *assistive* modifications aim to improve the experience of using the style. It is not easy to determine the exact effort that would need to be expended to implement the modifications suggested and so each modification is placed into one of three time scales in which it is believed each could be achieved, these are *weeks*, *months* and *years*. Table 7.1 groups all items of future work presented in this chapter in terms of their estimated value and time to perform.

7.1 Style Related

7.1.1 Static Properties

During the analysis of the enhanced style a number of properties were identified as areas for potential improvement:

Connector Creation and Destruction the characteristics here describe which participants in a connection have the privileges to create or destroy that connector. It may be the case however that those privileges are not static but are instead dependant on the state of the component or point in the conversation taking place. A means for modelling each components assumptions about states a conversation may adopt in such a way that allows models from different administrative domains to be compared could allow the creation and destruction characteristics to be more realistic.

	Substantive	*Assistive*
Weeks	• False Commission • Explicit Data Mapping In Messages Exchanged	• CSP Unparsed • Unique Process and Message Names • Case Sensitive Analysis Code • Exception Garbage Collection • Empty String Test • Sends First/Receives First • Mismatch Reporting
Months	• Data Continuity • Connector Creation and Destruction • Omission Mismatch Pessimistic • Commission and Omission, Fault or Failure? • Multiple Component Threads • Multiple Workflows • Loop Bounding • BPEL • Overlapping Choice Groups • Characteristic Publication	
Years	• Data Semantics • Failure Modes • Number of Traces Returned • Missing Properties	

Table 7.1: The estimated value and time to perform each of the items of future work discussed in this chapter.

Again this would represent a **substantive** improvement on the current situation, allowing a more realistic representation of when components expect to create and destroy connections therefore increasing the accuracy of the mismatch detection. It is envisaged that suitable models for describing this property and the associated analysis could be achieved on a time scale of **months**.

Data Continuity this characteristic is represented as an enumeration with two polar values, *sporadic* and *continuous* without them having any rigorous specification. This leaves this characteristic open to interpretation and weakens the single mismatch analysis result based upon it. This characteristic would therefore benefit from a more precise definition, one possible basis for this description would be to make explicit the Time Bands [BB05] that each component is using as its point of reference.

This would be a **substantive** improvement to the work as it would increase the expressiveness of the property and therefore potentially the accuracy of the mismatch result. It is envisaged that a suitable model for this property and the associated analysis could be achieved on a time scale of **months**.

Data Semantics the semantics in this work are simply represented as strings, and data semantics are compared by performing a string comparison, with string equality indicating semantic equivalence. This does not allow for any subsumption to be taken into consideration. For example, a car is a type of vehicle, in the current style this semantic similarity would be lost as the two strings are not equal. The future work here then would be to consider the use of an ontology language such as OWL [W3C09] to allow for such relationships to be better described. The addition of semantic relationships could allow for a greater range of mismatch results indicating the degree of semantic separation between two concepts.

The implementation of this item would improve the validity of the work by allowing the use of globally accepted or domain specific ontologies to determine the conceptual closeness of data items exchanged and so it is categorised as **substantive**. At the same time its reliance on both the generation of the ontologies and the means to determine how close concepts in those ontologies put it firmly in the **years** category for development time.

Failure Modes the failure modes exhibited and expected are chosen from the set of service failure modes presented by Avizienis *et al.* [ALRL04]. The question is whether these are sufficient and appropriate for an architectural description, or too abstract. If the latter is the case then what would be the next level of refinement. Also, the failure modes are considered currently on a point-to-point basis only, so each pair of connected ports must have compatible assumptions regarding the failure modes that may occur between them. This ignores the possibility of the failure being handled correctly by some other component in the system. For example, if a system consists of three components, A, B and C connected in a chain and A sends a message containing a content failure to B, then even if B cannot handle or detect the failure C may be able to, so the system as whole has some protection in the face of this type of failure. The addition of a model of failure handling and propagation could lead to a more realistic view of the system's fault tolerance as a whole.

The ability to determine how failures would propagate through a system would certainly be a **substantive** improvement on the current situation, but it is envisaged that it would require a considerable effort to achieve, putting it into the **years** category for effort.

7.1.2 Model Checked Properties

There are also a number of improvement opportunities relating to the description and analysis of the dynamic properties within the style:

Omission Mismatch Pessimistic The nature of the model checking used to detect omission mismatches means that the expected sender of a missing message is not known. This means the analysis can only know whether the receiving port is `inOurControlDomain` or not when determining if there is an omission mismatch or omission partial match. The other port is then assumed not to be within our domain of control. If the port that should have sent the message was known then its domain of control could be found allowing the mismatch to be correctly evaluated. With a change to the analysis source code, this could be achieved for those cases where there is only a single connector attached to the port experiencing the omission, however it is not currently possible where multiple connectors are attached. A means for handling with these latter cases would improve accuracy.

This improvement would increase the accuracy of mismatch reporting, its main benefit being that it would remove any time lost by the architect attempting to correct the mismatch by altering the wrong component. This modification is placed in the **substantive** category rather than assistive as the time spent trying to correct such a mismatch by altering the wrong component could be considerable. In terms of time it is hoped that a method for its implementation could be determined on the **months** scale.

Commission and Omission, Fault or Failure? Faults in systems lead to error states and if these are not handled they emerge from a component at which point it is said that a failure has occurred [AL81]. Commission and omission mismatches are detected when the model checking finds that an extra message is sent or an expected message is not received, these events along with the traces that lead up to them are then reported. However in both cases it may be that the commission or omission event is simply the result of following a branch in the conversation where the decision to follow that branch was made several steps earlier. In these cases, commission and omission may be described as failures of the system rather than a fault within it, the actual fault being the ability to follow the conversation branch leading to that point. The future work here then would be to support walking back along the trace to find the decisions made to reach this point and in doing so reveal the actual fault.

As above, the main benefit of this improvement is to reduce the time an architect might spend in tracking down the root cause of the mismatch, so again this is classified as being **substantive** and on the **months** time scale.

Multiple Component Threads Mismatch cc5 in Chapter 5 required that the number of concurrent threads present in a component be monitored. While the port CSP templates were modified so that concurrent threads in an individual port could be detected this does not yet support the detection of multiple threads in an individual component. The detection of the potential for a component to experience multiple threads spread accross a number of ports becomes important if those ports share a resource that is not protected against concurrency related problems such as race conditions. Additional modifications of the port CSP templates may allow this mismatch to be detected.

The style is currently unable to detect the presence of this mismatch type so its definition of the properties and related analysis would certainly constitute a **substantive** improvement. It is envisaged that, as was observed during the development of many of the CSP part of this work, a modelling solution and analysis could be determined on the time scale of **months**.

Multiple Workflows The port CSP templates all specify the next process or port to be followed once a port's message exchange pattern is complete. The result of this is that if there are two or more conversational threads within a component that make use of a common port, then these work flows are forced to exhibit the same behaviour after passing through the common port. A means for separating out the work flow from the port CSP templates while maintaining the "cooperative choice" principle would help in this respect.

The addition of multiple workflow support would increase the expressiveness of the style and so would constitute a **substantive** improvement over the choreographies currently possible using the style. Again it is envisaged the modelling and analysis solution could be determined in **months**.

Loop Bounding The current means for defining loops within the style does not support any bounding on the number of iterations. This might be supported using a separate work flow, as required above, to constrain the number of iterations performed.

As above, this would improve the expressiveness of the process modelling and so is also categorised as **substantive** and on the **months** time scale.

BPEL A number of the works cited in Chapter 2 relate themselves to the business process language BPEL [JE07] and the choreographies it is capable of expressing. Any future work on describing the conversations expected by a component should also be performed in the light of, and assessed against, this language to gain confidence in the completeness of the choreographic assumptions the style can express.

This would be a **substantive** addition to any work due to the confidence it could garner. It is reasonable to expect that acquiring the required understanding of BPEL and then performing a comparison of what choreographies the style allows against what BPEL supports could be completed in **months**.

7.1.3 Style Implementation

The final group of future work possibilities all stem from the approaches taken and decisions made during the implementation of the style and its analysis.

False Commission The existence of a false commission result stems from the commission and omission analyses not differentiating between messages sent by a component and those received. Correction of this simplification would remove this false result.

The correction of the external analysis to remove this issue would yield an improvement in the accuracy of the mismatch results returned and so it is listed as **substantive**. The underlying fault causing this issue is known and was discussed while demonstrating the stubborn connectors, Section 6.3.4, as such the implementation could be performed on the **weeks** time scale.

Explicit Data Mapping In Messages Exchanged The data in messages sent and received by a pair of ports are matched automatically based upon their declared semantics, once matched they can then have their data types and state scope assumptions compared. An alternative to this would be to manually describe the user's intended message data mappings in the connector. This would make the mappings explicit, an improvement on the current system where the mappings upon which the analysis is performed are not revealed in their entirety to the user. This would, of course, be at the cost of extra time declaring the mappings. A second option that could reduce the time cost would involve automatically generated data mappings that are revealed to the user within the tool environment for confirmation or adjustment. The first option of manual data mapping is possible currently within ACME Studio while the level of interaction between the user and the analysis rules required by the second option does not appear to be supported by ACME Studio at this time.

Either of the above options would constitute a **substantive** improvement to the style as they both reveal information that is currently hidden and either one could be implemented in **weeks** requiring only modifications to the style and/or modification to the related external analysis.

Overlapping Choice Groups The analysis code associated with multiple connections to a port assumes that each port will only be a member of a single choice group. It is conceivable that in a system with diverse work flows there will be ports that are members of multiple groups. A means for both representing and modelling such situations for analysis would be required to support this.

The addition of this feature would increase the expressiveness of the style and allow the representation of choreographies that the current CSP cannot, therefore it should be considered **substantive**. It would require modification of the CSP models and related external analysis so it is envisaged that it could take **months** to design and implement.

Number of Traces Returned A number of comments were made during the evaluation of this work that the model checker used was not returning all deadlock and refinement traces that one might expect of a model. This is despite, apparently, supplying the command line interface with the parameters for it to return the first 100 examples. While it is not possible to define a required number of traces and guarantee capturing all examples, receiving 100 would have allowed many more omission and commission examples to be reported by the analysis. A means for obtaining a more complete set of traces should be investigated.

This improvement is unique as its implementation is outside the control of the author, potentially requiring a modification to the FDR model checking tool itself. This makes the required effort an unknown, so a worst case is assumed and it is placed in the **years** category. Its implementation would certainly yield benefits to the analysis as the architect would be presented with a more complete view of the mismatch situation and so may be better able to choose the best correction strategy. This item is therefore listed as **substantive**.

CSP Unparsed The CSP descriptions are included as a string property in the model and are not parsed for syntax errors by either ACME Studio, which offers no such facility, or by the analysis itself. Parsing of the CSP would allow for constructive comment to be included in the exceptions output by the external analysis when the analysis is unable to construct a model, rather than attempting to process a flawed model and returning an unprocessed exception as is currently the case.

This does not add to the expressiveness of the style and so is an **assistive** item. Its implementation as an external analysis plug-in could be achieved in a matter of **weeks**.

Unique Process and Message Names The process and message names used in the CSP must be guaranteed unique by the user of the system. This task could, however, be automated by processing the CSP descriptions during construction of the system models.

As above, this modification is **assistive** in nature and a CSP pre-processor to ensure all process names were unique could be implemented on the **weeks** time scale.

Case Sensitive Analysis Code ACME Studio allows a characteristic to be declared in a model where the identifier string differs in terms of case from the characteristic declaration in the style. The interfaces provided within ACME Studio, that the external analysis use, however are very much case sensitive. The case sensitivity could be reduced by handling certain exception types when accessing the ACME model and performing the string comparisons after conversion to lower case.

The removal of the case sensitivity of the external analysis is an **assistive** improvement and could be performed on the order of **weeks**.

Exception Garbage Collection Much of the external analysis is dependant on an ACME model being completely defined, otherwise exceptions are raised. These exceptions exist as files and are not currently cleared once the exception no longer exists. Automating this would give confidence that a model is correctly specified.

This is a purely **assistive** improvement as the same effect can be achieved by the author deleting the exception files periodically, it could be implemented in within **weeks**.

Empty String Test In several rules the statement `X!=''''` is used to confirm that a string property is not empty. This was carried over from the minimal style, developed in ACME Studio 2.2.9b where it worked, and was reused in the enhanced style that was developed in ACME Studio 3.2.0 where it does not capture empty strings. An alternative approach should be found to confirm string properties are populated.

If we assume that the problem is not corrected by the developers of ACME Studio itself, then an **assistive** external analysis function to perform the same check could be implemented within **weeks**.

Mismatch Reporting The external analysis currently outputs any details regarding a discovered mismatch into a text file whose name is derived from the element ID and mismatch rule that was not passed. A potentially more convenient way of reporting this information would be to employ a view within ACME Studio, which after all is based upon Eclipse and supports plug-ins, and display the mismatch details there.

This would not reveal new information to the architect but would make it potentially more convenient to find, it is therefore **assistive** and could be implemented in **weeks**.

Sends First/Receives First These port properties are a part of the message exchange pattern analysis that is retained from the minimal style and was incorporated into the enhanced style analysis. However since the enhanced style now includes an explicit message exchange pattern identifier at the head of the port CSP, these are now redundant and should be removed from both the Armani and External analysis.

The removal of these redundant properties can be at most **assistive** as it would save the time needed to populate them. It is near trivial to implement and could be performed well within **weeks**.

7.2 SOA Related

7.2.1 Characteristic Publication

There are two parts to the publication of characteristics that would be of interest following this work.

The first would consider if and where the characteristics presented are available in the numerous web service description languages and, if they are, in what form do they exist.

The second part is based upon the assertion implicit in this work that these characteristics are important for detection of mismatches at composition time and so should be made explicit in the standard description of a web service component. From this, a study of where to and how to include these characteristics in a future version of WSDL could be of value.

This would be a **substantive** addition to the work as it may guide the architect to find the information required by the style that is currently not included in the WSDL specification. The development of a map of where the data could be found would require the examination of the many types of document that may be used to describe various aspect of web service components and so might be completed in a matter of **months**.

7.2.2 Missing Properties

A number of system characteristics were conspicuous by their absence from the works on architectural styles used as sources of properties to include. Most notable of these were the various dependability characteristics such as security, availability, reliability etc. Security is touched upon in the style in the form of the state scope assumptions, but this is a small part of security at best. Inclusion of such characteristics would greatly increase the coverage provided by the style presented and in doing so give a greater confidence in the composed system. It may be the case though that an application component may not specify its quality of service requirements, these instead might be specified elsewhere, perhaps as part of a contract between a service provider and client.

If such characteristics were added to the style along with the required analysis then this would surely be a **substantive** improvement, yielding a wider variety of mismatches than the current style supports. The definition of the characteristics of these properties, many of which could be considered non-functional is an open problem [PF10] and so is put firmly in the **years** time scale.

Chapter 8

Conclusions

This thesis set out to answer three central questions relating to architectural mismatches and SOA. Firstly, *Is the stipulated description of Web Service components sufficient to allow detection of all relevant architectural mismatches?* In this respect Chapter 3 showed that the minimum description required of Web Service components does allow some mismatches to be detected but comparing these to the mismatches identified in Chapter 4 reveals that it certainly does not allow detection of all relevant mismatches.

The second central question then asks, *If not, then what properties should both the services and the clients that use them make explicit to allow all relevant mismatches to be discovered?* The mismatches identified in Chapter 4 were used to drive the development of an Enhanced Web Service Architectural Style in Chapter 5. This style addresses the question by providing descriptions of client, intermediary and service components including all the properties and analysis required to allow detection of the majority of the identified mismatches.

The final central question focussed on the means for representing the systems and performing the analysis, *are architectural styles a suitable approach to support the represention and analysis of Web Service compositions for mismatch discovery?*. In this respect the architectural style acquitted itself well in the roles of providing guidance regarding the properties to be considered and then reporting on the results of the analysis. If a number of the suggestions in Chapter 7 were implemented then one could imagine such a system forming a useful tool for a Web Service composition process.

8.1 Key Contributions

The key contributions of this thesis can be summarised as follows:

Mismatches this work has presented two sets of mismatches. The first set are those that can be detected using only the standard WSDL document. The second set was derived from the architectural styles community literature. From this we were able to see the areas in which WSDL falls short.

Representation the enhanced architectural style presented in this work included example representations of the characteristics required to detect the above mismatches. These range from simple strings and enumerated sets to templates allowing the use of the CSP formalism to depict the conversational expectations of a component.

Detection to accompany the characteristics, the means for detection of each mismatch was presented with both a mathematical description and an example implementation in both ACME and Java where appropriate.

Demonstration both example scenarios and specially designed test system were used to demonstrate the effectiveness of the mismatch detection.

The contributions of this work show that while there is an overhead for the designer of each component, related to the additional characteristics they would need to populate in the component's description, there is a definite gain in terms of the scope of mismatches that can be detected. The demonstrated mismatches represent significant potential problems, such as the designer misinterpreting the semantics of the data exchanged in a message or the failure of a component to exchange messages as expected. Therefore it is suggested that the standard description documents for web services in particular or SOA in general need to take into account the properties proposed in this thesis, then tool vendors can consider including the analysis required to autonomously detect the mismatches during system composition.

8.2 Architectural Styles and Results

The thesis started by describing a Minimal Web Service Architectural Style in the ADL ACME and making use of the ACME Studio environment with its predicate language, Armani. This style included the significant properties available in a WSDL document and facilitated the representation of web service components architecturally. However it is not possible to detect architectural mismatches without also having some representation of the client components that use the services, so the style includes support for these and in doing so provides guidance to an architect regarding the properties to consider. A third type of component, an intermediary, is also included to represent those components described in the literature that offer a brokerage type service or mediate between incompatible components.

It was demonstrated during the evaluation that the style facilitates the detection of all those mismatches that can be made explicit using the minimal web service description.

The work then returned to the literature to determine a group of architecture characteristics deemed important for interoperability. This group was then reduced to reveal the subset that is significant in the scope of SOA. This resulted in some 20 separate mismatches found that could be relevant between an SOA component and its client.

The first use of these is to confirm the first thesis that "*It is not possible to detect, at configuration time, all architectural mismatches in a system comprising of web services given only the minimal web service description and specifications*". This is simply demonstrated by the mismatches found to be significant to SOA covering areas that the minimal style and WSDL do not touch upon.

The second use of the newly found mismatches, along with those highlighted from the work on the minimal architectural style was to drive the development of an architectural style that would facilitate their detection. The resulting Enhanced Web Service Architectural Style was certainly an improvement over the minimal style in terms of both representation of properties and coverage of mismatches detectable, however even this style did not detect all of the mismatches listed. The four mismatches that still cannot be detected are:

cp8 Mismatching state maintenance assumptions;

cc4 No component has an active thread of control;

cc5 Concurrent threads in single thread only component; and

cc7 Mismatching process distribution assumptions.

Of these it is believed that cp8 could be included if the architectural model contained a state view of the components, however ACME Studio does not support such a view. Cc7 could also be detected if a view mapping service components to physical hardware and networks were available.

An analysis rule could have been produced to detect a system where no component starts with an active thread of control, however this would be partially redundant as such a system would also report omission mismatches on all ports expecting to receive a message. Finally, it may be possible to detect cc5 with further development of the CSP templates used to represent the port message exchange patterns and the construction of an appropriate external analysis class.

One success of the enhanced style is its employment of templates to represent the message passing assumptions of the component ports and component threads of control in the process algebra CSP.

This allows the style to use the formal method along with the model checker, FDR, to detect mismatches caused by the emergent message passing behaviour of the system.

Based upon the above then it is not possible yet to confirm that the second thesis statement is true, though the hope is that with further development it could be possible to both describe the required properties and detect all the mismatches found relating to SOA. However, the enhanced style was still shown to have value by both detecting mismatches and then confirming their removal in a case study drawn from the literature.

While it is true that a small number of mismatches escaped the style and some of the analysis it includes could be improved in terms of the results returned, it does show that a style can be used, within a suitable environment such as ACME Studio, to detect mismatches and also provide a rigorous description of the properties an architect should consider when composing such a system.

8.3 Generalising

In more general terms, the style based approach worked well. It was found while building the examples that having a list of characteristics to populate allowed energies to be focussed on the task of deciding what values were appropriate rather than having to consider what properties should be included. While using ACME Studio it was found that the majority of time was spent creating the system model and populating the characteristics with values, the resulting analysis then taking very little time in comparison. This was partly due to all properties being manually populated, when in a more mature tool-kit one might reasonably expect to be able to import a complete Web Service description from perhaps some future version of WSDL or another service description language. Alternatively, if a component such as a client is being developed, then there may not be a complete description to import from. In such cases an improved user interface, possibly based upon the software wizard paradigm, could be employed to assist with the construction of the more complex data structures, such as the message definitions in the enhanced style.

While the styles presented in this thesis handled well the systems they were faced with, these systems were all constructed with the style in mind and did not, for example contain multiple styles like the pipe-and-filter and shared-data that exist in the example system in Figure 2.2 on page 15. If a style based approach were applied to such a system then it is at least plausible that the different styles employed may have contradictory specifications regarding individual characteristics. Careful design of the environment and possibly also the styles themselves, may be required to properly highlight such contradictions and also allow the suppression of warnings raised by whichever style constraints are ignored as part of the solution to the mismatch.

8.4 Reflections upon the Work

The author has heard it said that no PhD thesis is perfect, and this one is no exception. There now follow a short list of reflections upon the work performed in conducting this research.

The formalism CSP features very heavily in the choreographic properties and analysis in the style and for the most part it performs well. At the same time it is fair to say that the effort expended on adding a sort of state to the model to allow some of the analysis far outweighed the effort required to produce the basic CSP capable to representing the message exchanges between components. For example, it was necessary to determine methods to alter the basic CSP to support the detection of multiple threads in a port and also to ensure that messages were sent to the correct component when multiple components were connected to the same port. The efforts stem from CSP not having a natural mechanism for storing state.

One possibility for taking a different approach would be to have considered using a different formalism once the difficulties associated with using CSP became apparent. A distinct possibility for an alternate formalism would be Coloured Petri Nets (CPN) [Jen03]. CPNs are an extension of the standard Petri Net that allows the tokens to contain variables representing their state, these variables are known as the token's colour. Colouring could be used to record which component a message should be directed to if there are multiple components attached to a port. Colouring may also allow the implementation of multiple workflows for a component by using the colour to indicate which flow a token is following.

A second aspect of the work that would be altered with hindsight is the order in which the mismatches were tackled during the development of the enhanced style. The mismatches were attempted roughly in order of the assumed complexity of their representation and analysis. This left the more complex properties requiring the largest external analysis plug-ins till last. A PhD is a time limited project and so following the above method does not ensure that the highest value work is performed before the time expires, indeed it is only by virtue of the author securing a research position that the choreographic aspects were given more than a token treatment. The alternative approach would have seen the properties ranked according to their complexity and potential value to the project such that high value mismatches, in terms of their interest and contribution, could be attempted early on and the more trivial mismatches left till later.

The final reflection that will be made relates to the validation and motivation for the work. The subject of the thesis was derived from two sources, the author's previous masters research in the area of SOA and the supervisors' previous research on architectural mismatch. This resulted in work that had interest for both parties and was a pleasure to work on, but it meant that the work was not initiated by a concrete motivating example. The examples cited in Section 2.2.3 all relate to the problem of architecture mismatch but the literature did not yield documented examples of it occurring in the domain of web services. So while this work shows that the problem is possible in the web service domain, an actual concrete example would make for a more convincing case and would also help when describing the research to colleagues.

The lessons learned by the author from the above reflections could be summarised as

- Don't commit a path until it is necessary to, try to keep options and implementation details open
- Plan work according to the value it will return
- Work is ideally based on concrete examples

8.5 Final Conclusions

The overarching conclusions of this work are:

- The basic description of Web Service components is lacking important properties that are required to employ them with confidence;
- Client components also need to have explicit descriptions if compositions are to be analysed for mismatches;
- An architectural style can provide the support needed to detect mismatches and, if coupled with tools such as ACME Studio together with some of the suggested interface improvements, could form a valuable part of a Web Service composition tool kit;
- The enhanced web service architectural style itself provides extensive definitions of the properties required, for client, intermediary and service components, to permit mismatch detection by the analysis also described within this work. Additional investigation into the missing properties highlighted in the Future Work chapter can only serve to increase the confidence gained by employing the style to assess a SOA system composition for architectural mismatch.

Chapter 9

Glossary

ADL Architecture Description Language

Architectural Mismatch A situation where software components in a system make different and incompatible assumptions about the system they will be in

ASCII American Standard Code for Information Interchange

BPEL Business Process Execution Language

CBSE Component Based Software Engineering

Component A software component is a locus of computation and or storage in a system.

Connector A connector provides a conduit through which data and/or control may flow between components

Configuration A specific set of components, connectors their properties and the topology they form.

COTS Commercial Off The Shelf. A term given to software components purchased as is, without specialisation for the buyers purpose

CPN "Coloured Petri Net". A modification of Petri Nets that allow the tokens to contain state variables.

CSP "Communicating Sequential Processes". A process algebra for describing patterns of interaction between systems.

EFA Extended Finite-state Automata

FDR A CSP model checking tool produced by Formal Systems (Europe) Ltd. `http://www.fsel.com/software.html`

FSP Finite State Process

GUI Graphical User Interface

HTTP HyperText Transfer Protocol

INO In-Only - message exchange pattern

IOO In-Optional-Out - message exchange pattern

LTS Labelled Transition System

MSC Message Sequence Charts

NOTI Notification - message exchange pattern

OOI Out-Optional-In - message exchange pattern

Port A port represents an interface through which a component may exchange data and/or control with others

QoS Quality of Service

REQR Request-Response - message exchange pattern

RIO Robust-In-Only - message exchange pattern

Role A role is an endpoint of a connector, it attaches to a port to allow data and/or control to flow accross the connector

ROO Robust-Out-Only - message exchange pattern

RPC Remote Procedure Call

SENSORIA Software Engineering for Service-Oriented Overlay Computers[1]

SLA Service Level Agreements

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol. The protocol most commonly used by web services to format their messages, it uses XML as its basis. (It is also sometimes termed Service Oriented Architecture Protocol).

SOLI Solicit-Response - message exchange pattern

SRML SENSORIA Reference Modelling Language

[1] `http://www.sensoria-ist.eu/`

UDDI Universal Description, Discovery and Integration. A registry where clients may search for services by type and recive addresses of the WSDL descriptions of the service so they may bind to and use that service.

UML Unified Modeling Language

W3C World Wide Web Consortium

WS-I Web Services Interoperability Organisation

WSDL Web Service Description Language. An XML interface description laguage for web services. It defines any operations provided by a service and also any it may require in essentially terms of the messages sent and received by operation and the data types included in those messages along with a binding to an address, transport protocol (usually HTTP) and message encoding protocol (usually SOAP) to be used.

XML eXtensible Markup Language.

Bibliography

[AA96] A. Abd-Allah. *Composing Heterogeneous Software Architectures*. PhD thesis, Univeristy of Southern California, Los Angeles, CA, 1996.

[AL81] Tom Anderson and Peter A. Lee. *Fault Tolerance, Principles, and Practice*. Prentice-Hall, 1981.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004.

[ASAA09] Idir Ait-Sadoune and Yamine Ait-Ameur. A proof based approach for modelling and verifyingweb services compositions. In *ICECCS '09: Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems*, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society.

[Bak02] Sean Baker. Web services and corba. In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*, pages 618–632, London, UK, 2002. Springer-Verlag.

[BB05] Gordon Baxter and Alan Burns. Time bands in systems structure. In Denis Besnard, Cristina Gacek, and Cliff B. Jones, editors, *Structure for Dependability*, pages 74–88. Springer, 2005.

[BCK98] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

[Beh03] A. Behr. Defining the soa. *Software Development Times*, pages 29–31, November 1 2003.

[BEJV96] P. Binns, M. Engelhart, M. Jackson, and S. Vestal. Domain-specific software architectures for guidance, navigation, and control. *Int'l J. Software Eng. and Knowledge Eng.*, 6(2), 1996.

[BJPW] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making components contract aware.

[CBB+04] Paul Clements, Felix Bachman, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Juidth Stafford. *Documenting Software Architectures*. Addison Wesley, 2004.

[Che08] Yuhui Chen. *WS-Mediator for Improving Dependability of Service Composition*. PhD thesis, School of Computing Science, University of Newcastle, UK, 2008.

[CN08] Luca Cavallaro and Elisabetta Di Nitto. An approach to adapt service requests to actual service interfaces. In *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 129–136, New York, NY, USA, 2008. ACM.

[Col04] J. W. Coleman. Features of bpel modelled via structural operational semantics. Master's thesis, Newcastle University, 2004.

[DeL99] Robert DeLine. A catalog of techniques for resolving packaging mismatch. In *SSR '99: Proceedings of the 1999 symposium on Software reusability*, pages 44–53, New York, NY, USA, 1999. ACM Press.

[DeL01] Robert DeLine. Avoiding packaging mismatch with flexible packaging. *IEEE Transactions on Software Engineering*, 27(2):124–143, 2001.

[DGP02a] L. Davis, R. F. Gamble, and J. Payton. The impact of component architectures on interoperability. *J. Syst. Softw.*, 61(1):31–45, 2002.

[DGP02b] L. Davis, R. F. Gamble, and J. Payton. The impact of component architectures on interoperability. *J. Syst. Softw.*, 61(1):31–45, 2002.

[EK03] Amnon H. Eden and Rick Kazman. Architecture, design, implementation. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 149–159, Washington, DC, USA, 2003. IEEE Computer Society.

[FLB06] José Luiz Fiadeiro, Antónia Lopes, and Laura Bocchi. A formal approach to service component architecture. In *WS-FM*, pages 193–213, 2006.

[FS02] Kimiyuki Fukuzawa and Motoshi Saeki. Evaluating software architectures by coloured petri nets. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 263–270, New York, NY, USA, 2002. ACM.

[FS05] D. F. Ferguson and M. L. Stockton. Service-oriented architecture: programming model and product architecture. *IBM Syst. J.*, 44(4):753–780, 2005.

[FUMK03] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. *Automated Software Engineering, International Conference on*, 0:152–161, 2003.

[Gac98] Cristina Gacek. *Detecting Architectural Mismatches During Systems Composition*. PhD thesis, University of Southern California, 1998.

[Gam07] Carl Gamble. A minimal web service architectural style. Technical Report CS-TR-1015, Newcastle University, Newcastle upon Tyne, United Kingdom, 2007.

[GAO95] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch: why reuse is so hard. *Software, IEEE*, 12(6):17–26, 1995.

[Gro06a] ABLE Group. `http://www.cs.cmu.edu/~acme/`, 2006.

[Gro06b] ABLE Group. `http://www.cs.cmu.edu/~acme/AcmeStudio/index.html`, 2006.

[GS93] David Garlan and Mary Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pages 1–39, Singapore, 1993. World Scientific Publishing Company.

[HGK+06] M. Hepner, R. Gamble, M. Kelkar, L. Davis, and D. Flagg. Patterns of conflict among software components. *J. Syst. Softw.*, 79(4):537–551, 2006.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[JE07] Diane Jordan and John Evdemon. Web services business process execution language version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, April 2007.

[Jen03] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use (Volume 1)*, volume 1 of *EATCS Series*. Springer Verlag, April 2003.

[Joh05] C. W. Johnson. The natural history of bugs: Using formal methods to analyse software related failures in space missions. In J. S. Fitzgerald, I. J. Hayes, and A. Tarlecki, editors, *Formal Methods Europe 2005, LNCS 3582*, pages 9–25, 2005.

[Jon81] C. B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, June 1981. Printed as: Programming Research Group, Technical Monograph 25.

[KG98] R. Keshav and R. Gamble. Towards a taxonomy of architecture integration strategies. In *ISAW '98: Proceedings of the third international workshop on Software architecture*, pages 89–92, New York, NY, USA, 1998. ACM.

[McI69] M. D. McIlroy. Mass produced software components. *Software Engineering,NATO Science Committee*, pages 138–155, January 1969.

[MDT07] Nenad Medvidovic, Eric M. Dashofy, and Richard N. Taylor. Moving architectural description from under the technology lamppost. *Inf. Softw. Technol.*, 49(1):12–31, 2007.

[MG96] Robert T. Monroe and David Garlan. Style-based reuse for software architectures. In *ICSR '96: Proceedings of the 4th International Conference on Software Reuse*, page 84, Washington, DC, USA, 1996. IEEE Computer Society.

[MKMG97] Robert T. Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, January 1997.

[ML05] Leszek A. Maciaszek and Bruc Lee Liong. *Practical Software Engineering A Case Study Approach*. Addison Wesley, 2005.

[MMR06] Lee Momtahan, Andrew Martin, and A. W. Roscoe. A taxonomy of web services using csp. *Electr. Notes Theor. Comput. Sci.*, 151(2):71–87, 2006.

[Mon01] Robert T. Monroe. Capturing software architecture design expertise with armani. Technical Report CMU-CS-98-163, Carnegie Mellon University School of Computer Science, January 2001.

[MT00] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26(1):70–93, 2000.

[Nak05] Lightweight formal analysis of web service flows. *Progress in informatics : PI*, 2:57–76, 2005.

[Nak06] Shin Nakajima. Model-checking behavioral specification of bpel applications. *Electronic Notes in Theoretical Computer Science*, 151(2):89 – 105, 2006. Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005).

[NAS99] NASA. Mars climate orbiter mishap investigation board phase I report. `ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf`, 1999.

[OAS06] OASIS. Reference model for service oriented architecture v 1.0. `http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf`, August 2006.

[PA06] Shari Lawrence Pfleeger and Joanne M. Atlee. *Software Engineering, Theory And Practice, Third Edition*. Pearson Education, 2006.

[Pap08] Michael P. Papazoglou. *Web Services: Principles and Technology*. Prentice Hall, 2008.

[PF10] R. Payne and J.S. Fitzgerald. Evaluation of architectural frameworks supporting contract-based specification. Technical Report CS-TR-1233, Newcastle University, December 2010.

[Pum99] David John Pumfrey. *The Principled Design of Computer System Safety Analyses*. PhD thesis, Department of Computer Science, University of York, UK., 1999.

[PW92] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, 1992.

[PW05a] Savas Parastatidis and Jim Webber. Csp ssdl protocol framework. Technical Report CS-TR-901, School of Computing Science, Newcastle University, UK, 2005.

[PW05b] Savas Parastatidis and Jim Webber. Mep ssdl protocol framework. Technical Report CS-TR-900, School of Computing Science, Newcastle University, UK, 2005.

[PW05c] Savas Parastatidis and Jim Webber. The soap service description language. Technical Report CS-TR-899, School of Computing Science, Newcastle University, UK, 2005.

[PWW+05] Savas Parastatidis, Jim Webber, Simon Woodman, Dean Kuo, and Paul Greenfield. An introduction to the soap service description language. Technical Report CS-TR-898, School of Computing Science, Newcastle University, UK, 2005.

[SC96] M. Shaw and P. Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems, 1996.

[SC97] Mary Shaw and Paul C. Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. In *COMPSAC '97: Proceedings of the 21st International Computer Software and Applications Conference*, pages 6–13, Washington, DC, USA, 1997. IEEE Computer Society.

[Sch00] Steve Schneider. *Concurrent and Real-time Systems The CSP Approach*. WILEY, 2000.

[SG96] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, April 1996.

[Sha95a] Mary Shaw. Architectural issues in software reuse: It's not just the functionality, it's the packaging. In *ACM SIGSOFT Symposium on Software Reusability*, pages 3–6, 1995.

[Sha95b] Mary Shaw. Comparing architectural design styles. *IEEE Software*, 12(6):27–41, 1995.

[Som01] Ian Sommerville. *Software Engineering, Sixth Edition*. Addison Wesley, 2001.

[Sta06] Michael Stal. Using architectural patterns and blueprints for service-oriented architecture. *IEEE Softw.*, 23(2):54–61, 2006.

[Top03] Kim Topley. *Java Web Services In A Nutshell*. O'Reilly, 2003.

[UY00] Sebastian Uchitel and Daniel Yankelevich. Enhancing architectural mismatch detection with assumptions. *ecbs*, 00:138–147, 2000.

[VvdA05] H. M. W. Verbeek and W. M. P. van der Aalst. Analyzing bpel processes using petri nets. In *2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB 2005)*, pages 59–78. Florida International University, Miami, Florida, 2005.

[W3C06a] W3C. Soap version 1.2. `http://www.w3.org/TR/soap12/`, 2006.

[W3C06b] W3C. Web services architecture. `http://www.w3.org/TR/ws-arch/#introduction`, 2006.

[W3C06c] W3C. Web services description language 1.1. `http://www.w3.org/TR/wsdl`, 2006.

[W3C06d] W3C. Web services description language 2.0 part 0 : Core jan 6 2006. `http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106/`, 2006.

[W3C06e] W3C. Web services description language 2.0 part 1 : Core jan 6 2006. `http://www.w3.org/TR/2006/CR-wsdl20-20060106/`, 2006.

[W3C06f] W3C. Web services description language 2.0 part 2 : Adjuncts jan 6 2006. `http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060106/`, 2006.

[W3C09] W3C. Owl 2 web ontology language. `http://www.w3.org/TR/owl2-overview/`, 2009.

[YBB99] Daniil Yakimovich, James M. Bieman, and Victor R. Basili. Software architecture classification for estimating the cost of cots integration. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 296–302, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[Yeu06] Wing Lok Yeung. Mapping ws-cdl and bpel into csp for behavioural specification and verification of web services. In *ECOWS*, pages 297–305, 2006.

[YTX05] YanPing Yang, QingPing Tan, and Yong Xiao. Verifying web services composition based on hierarchical colored petri nets. In *IHIS '05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pages 47–54, New York, NY, USA, 2005. ACM.

[YWD06] W. L. Yeung, Ji Wang, and Wei Dong. Verifying choreographic descriptions of web services based on csp. In *SCW '06: Proceedings of the IEEE Services Computing Workshops (SCW'06)*, pages 97–104, Washington, DC, USA, 2006. IEEE Computer Society.

Appendix A

ACME Studio Introduction

ACME, Armani and ACME Studio are extensively used throughout this work, so an introduction is required. It will cover the language and its tool support, outlining the important features of each and providing example description fragments to help familiarisation.

A.1 ACME Architecture Description Language

ACME was designed as an architecture interchange language where interchange means a common language that tools designed for different ADLs could use to exchange data. It was designed from the ground up then to explicitly support the most basic architectural elements, *components, connectors, ports, roles*, and their structural relationships.

When building an ACME model the first step is to define the system, its name and reference any architectural styles it will employ, Figure A.1.

After that there is no prescribed order, but one approach is to define the components and their ports, the connectors and their roles and finally make the attachments, which is the point when the components become a system.

A component is defined by first declaring its name and type before adding any child elements and properties. The properties can be of any of the primitive types supported by ACME or may be a composite type defined in the architectural style.

The child elements of a component are its ports, these represent the interface it presents to the environment. Ports are declared within the description of the component itself and there is no restriction on the number of instances or types a component may have. Ports like components can

```
import families\ws_minimal_3.acme;
System exampleSystem : ws_minimal_3 =  new ws_minimal_3 extended with {
        \\ description of system elements goes here
};
```

Figure A.1: System declaration in ACME

```
Component SNP : CompTWSIntermediary =  new CompTWSIntermediary extended with {
    Port calcRoute : PortTWSService =  new PortTWSService extended with {
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property Interface : Interfaces = Service;
        Property EndPointAddressList : EndPointAddresses = {"snp.com/calcRoute"};
        Property SendsFirstMessage : SafeBoolean = No;
        Property InOurControlDomain : SafeBoolean = Yes;
        Property WsdlDocRefs : WsdlDocs = {"http://wsdl.snp.com"};
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "routeCriteria";
            DT = "out" ], [
            ST = "pathData";
            DT = "in" ] >, < [
            ST = "routeCriteria";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };
        // the rest of the ports in this component would be described here also
};
```

Figure A.2: A component description in ACME containing a single port with a number of properties

```
Connector ConnTWS1 : ConnTWS =  new ConnTWS extended with {
    Role r1;

    Role r2;
};
```

Figure A.3: Description of a connector with no properties, in ACME

also have properties but cannot have child elements, Figure A.2.

The other major architectural element is the connector. Like the components before it is a first class entity and so is declared at system level. Also like components it can have properties and child elements, which in this case are roles. In the same way that ports represent the interface of a component, roles represent the interaction points of a connector. Roles are declared within the description of a connector and may have their own properties but no child elements, Figure A.3.

The final step in building an ACME model is to attach the roles to the required ports to form the structure of the system, Figure A.4.

If required it is possible to refine the components and connectors into their constituent elements and structures to enable further development. ACME supports this in the form of *representations*. A representation is defined as a system in its own right and as such its description shares almost the same structure and the parent system it exists within. The only difference is that a representation is obliged to implement the interface presented by its parent element. Thus while many of the ports and roles in the representation are attached to each other, some of them are *linked* to the ports (or

```
Attachment SNP.calcRoute to ConnTWS1.r1;
Attachment SN.getRoute to ConnTWS1.r2;
```

Figure A.4: An example of attaching a connector `ConnTWS1` to ports on components `SNP` and `SN`

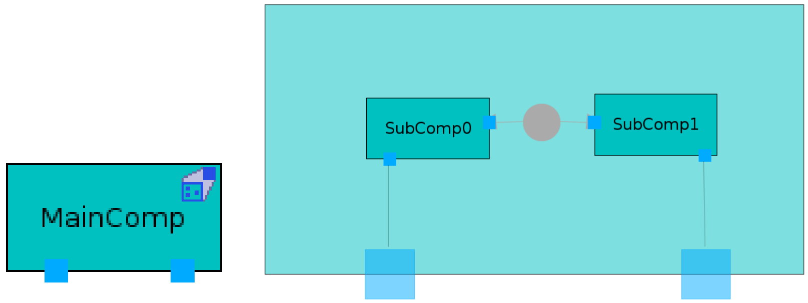


Figure A.5: Graphical view of a component names “MainComp”, shown on the left, with its internal representation shown on the right.

roles) of the parent component (or connector), Figures A.5 and A.6.

This allows description of the structure and properties of the architecture of a system, however to automatically analyse the system requires rules or constraints in the form of an architectural style. An ACME style is defined using a similar structure to a system, with the exception that the declaration now refers to types rather than instances and constraints can now be included.

A style description starts with a declaration of its name and any styles[1] it extends, Figure A.7.

To support being an interchange language ACME permits us to either use the primitive data types it supplies or build bespoke data structures using them. For example a CSP [Hoa85] description may be stored as a simple string type whereas an interaction between two ports could also be described by defining a message data type and a description of the message exchange pattern.

ACME provides a number of collection data structures, with which it is possible to build complex data types. Firstly there are **Sets**, which contain elements of a defined type with no duplicates allowed. **Enumerations** allow definition of a set of values of a type, which can then be used to describe the allowed values for properties. **Sequences** are ordered lists of elements of a defined type in which duplicates are allowed. Finally there are **Records**, which are structures which are defined to contain a number of individually named and typed values. For all of these data structures the types they hold can be any of the other collection structures, so a sequence of records is possible, which allows the construction of varied data types for use in the architecture descriptions.

Once the data types are in place the component types may be defined by naming the types and declaring what properties are expected within the types along with any default values. Any default child element instances may also be defined, Figure A.8.

The same then applies to the port, connector and role types that make up a style, for example a connector type declaration is shown in Figure A.9.

Already with this style achieves two things. Firstly it guides the architect with regard to what parameters are deemed to be important and secondly it allows the tool support to warn the architect

[1] In ACME parlance and architectural style is termed a “family” which the reader may notice in the description excerpts, however we will use the former term during this work

```
Component MainComp  = {
    Port Port0  = {...}
    Port Port1  = {...}

    Representation MainComp_Rep = {
        System MainComp_Rep = {

            Component SubComp0  = {
                Port Port0  = {...}
                Port Port1  = {...}
            }

            Component SubComp1  = {
                Port Port0  = {...}
                Port Port1  = {...}
            }

           Connector conn  = {
                Role r0  = {...}
                Role r1  = {...}
            }

            Attachment SubComp0.Port1 to conn.r0;
            Attachment SubComp1.Port0 to conn.r1;
        }

        Bindings {
            MainComp.Port0 to SubComp0.Port0;
            MainComp.Port1 to SubComp1.Port1;
        }
    }
}
```

Figure A.6: The representation shown in Figure A.5 in its ACME form.

```
Family ws_minimal_3 = {
        // style description goes here
}
```

Figure A.7: A style declaration in ACME, this style does not extend any others

```
Component Type CompTWSService extends CompTWSCommon with {

        // rule checking the component has at least one port
        invariant size(self.ports) > 0<<label : string = "Component has at least one port";
            errMsg : string = "Component should have at least one port";>>;

        PortTWSCommon Port0;
.          .          .
.          .          .
.          .          .

}

Port Type PortTWSCommon = {
        // Property that holds the "wire" protocols, i.e. transport and encoding
        // protocol pairs that this port supports
        Property EndPointList : EndPoints;

        invariant size(EndPointList) > 0<<label : string = "Endpoint list is populated";
            errMsg : string = "Endpoint list must be populated";>>;
.          .          .
.          .          .
.          .          .
}
```

Figure A.8: Declaring and component and port type in the style. Both contain invariant rules, which are described later. The component requires that a port of type `PortTWSCommon` be declared as a child element.

```
Connector Type ConnTWS = {
    // These connectors are currently prevented from providing multicast facilities,
    // a multicast can only be achieved by explicitly instantiating multiple
    // connectors
    invariant size(self.roles) == 2
        <<label : string = "A connector of this type must have 2 roles";
        errMsg : string = "This connector must have exactly two roles";>>;
};
```

Figure A.9: An example connector type declaration in a style.

```
invariant Forall r1 : role in self.roles |
  Forall r2 : role in self.roles |
    Forall p1 : PortTWSCommon in r1.attachedPorts |
      Forall p2 : PortTWSCommon in r2.attachedPorts |
        (r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
        size(intersection(p1.EndPointList, p2.EndPointList)) > 0
        <<label : string = "Ports have a matching Transport / Encoding pair";
        errMsg : string = "No matching pair of endpoint protocols";>>;
```

Figure A.10: A rule which, if in a connector, will select the two port instances the connector is attached to and will then evaluate the size of the intersection of their `EndPointList` property.

if the properties are either none existent or if they are of the wrong data type. This is really a syntactic check, however ACME also supports the inclusion of constraints described in the predicate language Armani which allows for more powerful checks to be performed on a model.

A.2 Armani Predicate Language

Rules written in the Armani predicate language have two main parts to them, selection and evaluation.

Selection is the process of finding the architectural elements of importance to the rule. Evaluation is a boolean function over those elements and their properties.

The location of the rule definition in the style description is significant as this defines the scope of that rule and sets the context from which the selections can be made. For example if a rule is defined inside a connector type called `TConnA`, then the rule will be invoked wherever a connector of that type is instantiated in a system. Also it will have its scope limited to those connector instances and the roles and ports directly attached to it. This means a rule can evaluate properties of the connector itself, its roles and the ports attached to it. This is achieved by traversing the sets provided by ACME, Figure A.10.

This scoping also means that the same rule can be evaluated for each instance of the connector and will return true or false (pass or fail effectively) dependant on the individual circumstances of each connector.

Rules can be defined with any level of scope depending on where they are declared. So a rule in a port definition can only “see” individual instances of that port. But a rule defined outside all the element type definitions will have global scope in a system and can evaluate all elements and their

```
    invariant Forall comp : component in self.Components |
        satisfiesType(comp, CompTWSClient)
     OR satisfiesType(comp, CompTWSService)
     OR satisfiesType(comp, CompTWSIntermediary)
     <<label : string = "All components are WSClients, WSServices or WSIntermediarys";
     errMsg : string = "Style only permits WSClient, WSService and WSIntermediary
                        type components";>>;
```

Figure A.11: A global rule, which exists in the root of the style description, this checks that all components in a system satisfy one of the types `CompTWSClient, CompTWSService` or `CompTWSIntermediary`.

```
heuristic Forall r1 : role in self.roles |
  Forall r2 : role in self.roles |
    Forall p1 : PortTWSCommon in r1.attachedPorts |
      Forall p2 : PortTWSCommon in r2.attachedPorts |
        (r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
        (!((p1.InOurControlDomain == Yes
            AND
            (!(isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns)))
            AND
            isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns)
           )
         OR
           (p2.InOurControlDomain == Yes
            AND
            (!(isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns)))
            AND
            isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns)
           )
         ))
```

Figure A.12: A complex rule using multiple logic statements to conditionally evaluate the two ports attached to a connector.

properties, Figure A.11. Finally, if an element type is defined as extending another type, then the new type will also inherit any rules the super type contained.

Once the required elements have been selected, boolean expressions are employed to evaluate properties of interest. These expressions include simple equalities of the property values, set operations such as checking for subsets[2] and existential functions among others [Mon01]. The normal boolean operators (And & Or) may also be used to construct more complex statements. The rule shown in Figure A.10 only contains a single evaluation statement, while the rule shown in Figure A.12 uses logic operators to perform a conditional evaluation of two ports.

A.3 External Analysis

While Armani allows for complex statements to be constructed it is still limited to a set of generic boolean functions, ACME Studio does, however, afford the user the opportunity to extend this set with their own external analysis.

The purpose of external analysis is to allow the user to define their own means for evaluating the elements and their properties. The external analyses are Java classes that are packaged as Eclipse

[2] the Armani expression `isSubset(A,B)` returns true if $A \subseteq B$, there is no proper subset expression other than negating the reverse.

```
Family externalAnalysisExample = {

     external analysis chkname(a : element) : boolean = uk.ac.ncl.CompNameCheck;

Component Type comp1 = {
                   Property name : String;
          rule checkComponentName = heuristic chkname(self);
     }
}
```

Figure A.13: An example of declaring an external analysis which uses a Java class (CompNameCheck in the package uk.ac.ncl) to perform the evaluation. This analysis is then used in the rule checkComponentName in the component type comp1.

plug-ins. The plug-ins return a boolean value indicating if a system or element passes or fails the analysis it represents, Figure A.13.

There are a number of features which make external analysis potentially more powerful and flexible than the Armani provided functions.

Firstly, being effectively Java programs in their own right they can have all the power and freedom of any Java application. This means arbitrarily complex analysis may be performed or the Java application may form a connector to existing application to use its functionality. Specifically in this work the FDR[3] CSP model checker is used in this way.

Secondly, external analysis is not constrained by rule definition scope in the same way as the normal Armani analysis. The java representations of the architectural elements provide methods to allow access to their parent elements, something which is not possible in Armani. This means that given a reference to a port object, it is possible to get access to its parent component object and from there get access to the system it exists in. Thus external analysis is able to traverse the entire architecture model reaching any element from any other element. Much of the analysis described in Chapter 5 would not have been possible without this feature.

A.4 ACME Studio and ACME Libs

ACME was chosen not only for its language but also for its tool support which is available in two distinct versions ACME Studio, the graphical editing and analysis tool , and ACME Libs, which are the underlying command line libraries and parser.

ACME Studio is built upon the Eclipse[4] platform so has a modular layout that users of that is split into 5 panes (Figure A.14). These have the following purposes :

Project Explorer this is the standard project explorer provided by Eclipse. In here the user can manage the files associated with a project, this is also one of the places to create new projects and the systems / styles within them.

[3]FDR CSP model checker, of Formal Systems (Europe) Ltd. http://www.fsel.com/software.html

[4]http://www.eclipse.org/

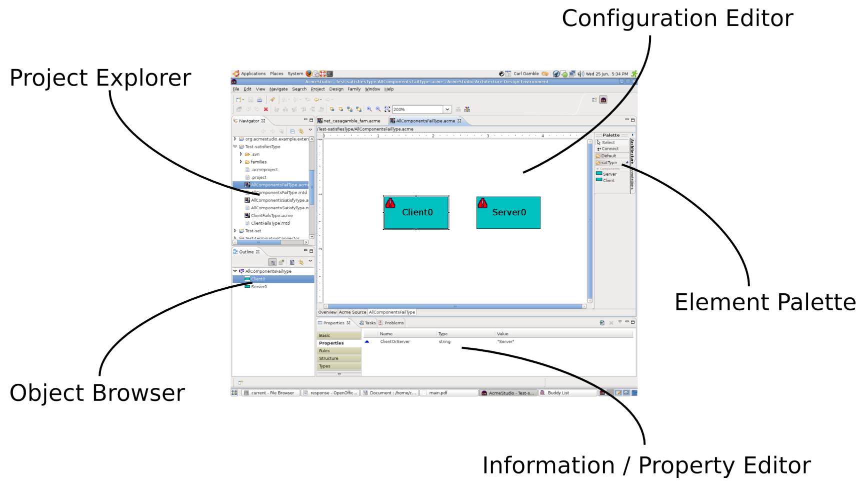


Figure A.14: ACME Studio basic layout, with the Configuration / Source / Style parameters editor shown in the configuration view.

Configuration / Source / Style Parameters Editor This pane is where most of the actual work is performed and it has three main functions :

1. It provides the graphical view of the system being developed and allows the architectural elements to be dragged and dropped from the element palette (described below). It also allows the elements to be positioned / resized to represent the system[5]. It is in this view that warning triangles appear to indicate which elements do not satisfy their type, that is one or more rules they contain do not evaluate to true or one or more properties are not defined or have invalid values.

2. The second view allows direct editing of the system or style description source directly. It provides simple highlighting of lines containing syntax errors upon the source file being saved.

3. The final view, only available in the context of editing an architectural style description. It allows the setting of visual parameters which are associated with the element types included in the style, such as colours and shapes.

   This is also where the "connection patterns" are defined. These are tuples of *component type - port type - role type - connector type - role type - port type - component type*. These patterns are used by the graphical editor to determine when to attach a role to a port. This feature does not relate to any topological constraints, if any, defined in the style itself, also attachments may be made in the source editor which do not relate to the patterns at all. None of the changes in this view save for creating or editing a type or its "actual" properties are reflected in the style at all.

[5]This has no effect on the semantics of the system, it is purely cosmetic.

Element Palette When in the graphical view (above) this presents a palette of all architectural element types available as standard in ACME Studio and those provided in any styles adopted by the system. It is from here that element types may be dragged and dropped to produce instances in the system.

Information / Property Editor This panel is where data specific to a selected element is displayed. It allows editing of the values of the properties of that element, which avoids editing them directly in the source pane. Most usefully though, it also lists all rules which apply to that element, the rule IDs and whether or not each individual is satisfied or not. This then gives the details of which rules caused a warning triangle to appear on the graphical display, this is vital to the success of this environment as a means to detect mismatches.

Object Browser This presents a hierarchical view of the system and all elements within it. Selecting an element here has the same effect as selecting it in the graphical view, but is sometimes easier as ports, roles and connectors are not labelled with their names in the graphical view.

ACME Libs is a much simplified command line means to use the libraries that underpin ACME Studio. It therefore has the same analytical abilities as its graphical sibling, but it provides no system editing facilities, providing instead only parsing and rule evaluation of an ACME model. This results in a verbose text output detailing each and every rule that is not satisfied.

It has two potential benefits over the ACME Studio tool.

1. the analysis runs once and once only on the model while ACME Studio continuously works through all analysis rules. The difficulty with ACME Studio comes from the silent execution of analysis rules, this means that after a change had been made to a system model the user does not know for sure when that change has been viewed by all rules[6].

2. it does not require the external analysis classes to be packaged as Eclipse plugins they can simply be Java classes.

[6]The approach taken in this work was to close and restart ACME Studio after changes were made to a model, then when all rules had been evaluated it was known that the results represented the current model and not an earlier state.

Appendix B

Minimal Style Description

```
Family ws_minimal_3 = {
    // Below are the custom types used in this style, the syntax does not allow them
    // to be defined in the connectors where the properties based upon them are
    // instantiated

        //This represents a set of strings which are intended to hold valid URIs to valid
        // WSDL documents
    Property Type WsdlDocs = Set{string};

    // A safe boolean type property. This allows us to check that a user has
    // populated it unlike a boolean, which if not initialised defaults to returning
        //true when queried.
    Property Type SafeBoolean = Enum { Yes, No };

    // Defines the set of legal soap versions as tokens, which are utilised in the
        // EndPoint type
    Property Type legalSoapVersions = Enum { SOAP1_1, SOAP1_2 };

    // Defines the set of legal transport protocols as tokens, this set is in no way
    // complete. The set is utilised in the TransportProtocols set
    Property Type legalTransportProtocols = Enum { HTTP1_0, HTTP1_1 };

    Property Type EndPoint = Record [
        Transport : legalTransportProtocols;
        Encoding : legalSoapVersions;
    ];

    Property Type EndPoints = Set{EndPoint};

    Property Type EndPointAddresses = Set{string};

    //The definition of a "message" type, a "validExchange" type and a
    // "messagePatterns" type, which can be used to define, using tokens, the
    // message exchanges a port can accept. The message is weakly defined as a token
    // representing the syntax of the message (ST) and a token representing its
    // direction (in, out), the direction is always defined from the point of view
    // of the port initiating the message exchange. i.e. the first message in a
    // valid exchange will always have DT = "out"
```

```
Property Type message = Record [
    ST : string;
    DT : string;
];

Property Type validExchange = Sequence<message>;

Property Type messagePatterns = Set{validExchange};

    // An enumerated type to distinguish ports which are intended to be part of the
    // client interface of a component, or its service interface.
Property Type Interfaces = Enum { Client, Service };

    // ***Below are the configuration rules***

// Checks that all components in the system satisfy the requirements of being a
// web service
invariant Forall comp : component in self.Components |
    satisfiesType(comp, CompTWSClient) OR
    satisfiesType(comp, CompTWSService) OR
    satisfiesType(comp, CompTWSIntermediary)
        <<label : string = "All components are WSClients, WSServices or WSIntermediarys";
        errMsg : string = "Style only permits WSClient, WSService and WSIntermediary
            type components";>>;

// Checks that all connectors in the system satisfy the requirements of being a
// web service type
invariant Forall conn : connector in self.connectors |
    satisfiesType(conn, ConnTWS)
        <<label : string = "All Connectors are WS type";
        errMsg : string = "Either a non web service connector has been used or a
            connection has been made which breaks one or more rules";>>;

// *** Below are the component types***

Component Type CompTWSCommon = {
}

Component Type CompTWSClient extends CompTWSCommon with {
    // Rule checking all associated ports conform to the Client port type
    invariant Forall p : port in self.Ports |
        satisfiesType(p, PortTWSClient)
            <<label : string = "External ports are all Client type";
            errMsg : string = "Only client type ports are allowed";>>;

    // rule checking the component has at least one port
    invariant size(self.ports) > 0
        <<label : string = "Component has at least one port";
        errMsg : string = "Component should have at least one port";>>;

}


Component Type CompTWSService extends CompTWSCommon with {
```

```
        // Rule checking all associated ports conform to the Service port type
        invariant Forall p : port in self.Ports |
            satisfiesType(p, PortTWSService)
            <<label : string = "External ports are all Service type";
            errMsg : string = "Only service type ports are allowed";>>;

        // rule checking the component has at least one port
        invariant size(self.ports) > 0
            <<label : string = "Component has at least one port";
            errMsg : string = "Component should have at least one port";>>;

    }


    //
    Component Type CompTWSIntermediary extends CompTWSCommon with {
        // Rule checking all associated ports conform to the Client or Service type
        invariant Forall p : port in self.Ports |
            satisfiesType(p, PortTWSClient) OR
            satisfiesType(p, PortTWSService)
            <<label : string = "External ports are of the web service type";
            errMsg : string = "Only WebService type ports are allowed";>>;

        // rules checking the component has at least one client port and one service
        // port
        invariant Exists p : port in self.Ports |
            satisfiesType(p, PortTWSClient)
            <<label : string = "Component has at least one client type port";
            errMsg : string = "Component must have at least one client type port";>>;

        invariant Exists p : port in self.Ports |
            satisfiesType(p, PortTWSService)
            <<label : string = "Component has at least one service type port";
            errMsg : string = "Component must have at least one service type port";>>;

    }
    // *** Below is the single connector type***

    Connector Type ConnTWS = {
        // These connectors are currently prevented from providing multicast facilities,
        // a multicast can only be acchieved by explicitly instantiating multiple
        // connectors
        invariant size(self.roles) == 2
            <<label : string = "A connector of this type must have 2 roles";
            errMsg : string = "This connector must have exactly two roles";>>;

        // Rule checking for at least one common end point protocol pair
        invariant Forall r1 : role in self.roles |
            Forall r2 : role in self.roles |
            Forall p1 : PortTWSCommon in r1.attachedPorts |
            Forall p2 : PortTWSCommon in r2.attachedPorts |
            (r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
                size(intersection(p1.EndPointList, p2.EndPointList)) > 0
                <<label : string = "Ports have a matching Transport / Encoding pair";
```

```
        errMsg : string = "No matching pair of endpoint protocols";>>;

// Part 1 of 2 of message passing rules : heuristic that flags a connection
// where only a partial match of message patterns is made, this is to warn that
// the calling services behaviour should be restricted to that compatible with
// the called service.
heuristic Forall r1 : role in self.roles |
    Forall r2 : role in self.roles |
    Forall p1 : PortTWSCommon in r1.attachedPorts |
    Forall p2 : PortTWSCommon in r2.attachedPorts |
    (r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
    (!
      (
        (p1.InOurControlDomain == Yes
         AND
        (!(isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns)))
         AND
         isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns)
        )
        OR
        (p2.InOurControlDomain == Yes
         AND
        (!(isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns)))
         AND
         isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns)
        )
      )
    )
    <<label : string = "Check for a full match";
    errMsg : string = "Services partialy compatible,
        behaviour of one service should be constrained!";>>;

// part 2 of 2 of message passing rules : invariant checking that there is
// either a partial or full match of the message patterns between the connected
// ports, otherwise raises an error highlighting incompatible ports.
invariant Forall r1 : role in self.roles |
    Forall r2 : role in self.roles |
    Forall p1 : PortTWSCommon in r1.attachedPorts |
    Forall p2 : PortTWSCommon in r2.attachedPorts |
    (r1 != r2 AND attached(r1, p1) AND attached(r2, p2)) ->
        (p2.MessageExchangePatterns == p1.MessageExchangePatterns)
        OR
        ( p1.InOurControlDomain == Yes
          AND
          (!(isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns)))
          AND
          (isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns))
        )
        OR
        ( p2.InOurControlDomain == Yes
          AND
          (!(isSubset(p2.MessageExchangePatterns, p1.MessageExchangePatterns)))
          AND
          (isSubset(p1.MessageExchangePatterns, p2.MessageExchangePatterns))
```

```
            )
            <<label : string = "Check for a partial match";
            errMsg : string = "Message exchange patterns or message signatures
                do not match";>>;

        invariant Exists r : role in self.roles |
            Forall p : PortTWSCommon in r.attachedPorts |
            attached(r, p) -> p.SendsFirstMessage == Yes
            <<label : string = "One port expects to send the first message";
            errMsg : string = "Neither port expects to send the first message";>>;

        invariant Exists r : role in self.roles |
            Forall p : PortTWSCommon in r.attachedPorts |
            attached(r, p) -> p.SendsFirstMessage == No
            <<label : string = "One port is listening for the first message";
            errMsg : string = "Neither port is listening for the first message";>>;

    }

    // *** Below are the port types***

    Port Type PortTWSCommon = {
        // Property that holds the "wire" protocols, i.e. transport and encoding
        // protolcol pairs that this port supports
        Property EndPointList : EndPoints;

        invariant size(EndPointList) > 0
            <<label : string = "Endpoint list is populated";
            errMsg : string = "Endpoint list must be populated";>>;

        // Property that determines if this port is within "our" domain of control and
        // "we" may be able to alter its behaviour
        Property InOurControlDomain : SafeBoolean



        invariant InOurControlDomain == Yes OR InOurControlDomain == No
            <<label : string = "In our control domain property is populated";
            errMsg : string = "In Our Control Domain property must be populated";>>;

        // placeholder for the message exchange pattern data, with a rule checking
        // that it is populated
        Property MessageExchangePatterns : messagePatterns;

        invariant size(MessageExchangePatterns) > 0
            <<label : string = "Message exchange pattern is populated";
            errMsg : string = "Message exchange pattern must be populated";>>;

        // does this port send the first message in an exchange or does it wait for the
        // first message to come in, followed by a rule checking it is populated
        Property SendsFirstMessage : SafeBoolean;

        invariant SendsFirstMessage == Yes OR SendsFirstMessage == No
            <<label : string = "Sends first message property is populated";
```

```
            errMsg : string = "Sends First Message property must be populated";>>;

    }


    Port Type PortTWSClient extends PortTWSCommon with {
        Property InInterface : Interfaces = Client;

    }


    Port Type PortTWSService extends PortTWSCommon with {
        Property InInterface : Interfaces = Service;

        // holds the list of endpoint addresses of this port
        Property EndPointAddressList : EndPointAddresses;

                // rule check the End point address list is populated
        invariant size(EndPointAddressList) > 0
            <<label : string = "Endpoint address list is populated";
            errMsg : string = "Endpoint address list must be populated";>>;

                // rule check there are as many end point addresses as there are end points
        invariant size(EndPointAddressList) == size(EndPointList)
            <<label : string = "Number EndPoint addresses = number of EndPoint protocol pairs";
            errMsg : string = "Must be one End Point Address for each End Point protocol pair";>>;

        // placeholder for the WSDL document references, with a rule checking each port
        // is referenced by at least one doc
        Property WsdlDocRefs : WsdlDocs;

        invariant size(WsdlDocRefs) > 0
            <<label : string = "WSDL reference list is populated";
            errMsg : string = "WSDL reference list should be populated";>>;
    }
}
```

Appendix C

Complete ACME Descriptions of Minimal Style Scenario

```
import $AS_PROJECT_PATH\families\ws_minimal_3.acme;
System SatNavScenario : ws_minimal_3 =  new ws_minimal_3 extended with {
    Component SNP : CompTWSIntermediary =  new CompTWSIntermediary extended with {
        Port calcRoute : PortTWSService =  new PortTWSService extended with {
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property InInterface : Interfaces = Service;
            Property EndPointAddressList : EndPointAddresses = {"snp.com/calcRoute"};
            Property SendsFirstMessage : SafeBoolean = No;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.snp.com"};
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "routeCriteria";
                DT = "out" ], [
                ST = "pathData";
                DT = "in" ] >, < [
                ST = "routeCriteria";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port checkStatus : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property EndPointAddressList : EndPointAddresses = {"snp.com/statusRequest"};
            Property SendsFirstMessage : SafeBoolean = Yes;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.snp.com"};
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "requestStatusAndLocation";
                DT = "out" ], [
                ST = "statusAndLocation";
```

```
                DT = "in" ] >, < [
                ST = "requestStatusAndLocation";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port updateRoute : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property EndPointAddressList : EndPointAddresses = {"snp.com/updateRoute"};
            Property SendsFirstMessage : SafeBoolean = Yes;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.snp.com"};
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "newPathData";
                DT = "out" ] >};
        };

        Port requestAssistance : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property EndPointAddressList : EndPointAddresses = {"snp.com/requestAssistance"};
            Property SendsFirstMessage : SafeBoolean = No;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.snp.com"};
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "requestAssistance";
                DT = "out" ], [
                ST = "assistanceOffers";
                DT = "in" ] >, < [
                ST = "requestAssistance";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port assistanceChoice : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property EndPointAddressList : EndPointAddresses = {"snp.com/assistanceChoice"};
            Property SendsFirstMessage : SafeBoolean = No;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.snp.com"};
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "assistanceChoice";
                DT = "out" ], [
                ST = "assistanceConfirmation";
```

```
            DT = "in" ] >, < [
            ST = "assistanceChoice";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

    Port assistanceUpdate : PortTWSService =  new PortTWSService extended with {
        Property InInterface : Interfaces = Service;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property EndPointAddressList : EndPointAddresses = {"snp.com/assistanceUpdate"};
        Property SendsFirstMessage : SafeBoolean = Yes;
        Property InOurControlDomain : SafeBoolean = Yes;
        Property WsdlDocRefs : WsdlDocs = {"http://wsdl.snp.com"};
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "updateOffer";
            DT = "out" ], [
            ST = "isUpdateAccepted";
            DT = "in" ] >, < [
            ST = "updateOffer";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

    Port requestOffer : PortTWSClient =  new PortTWSClient extended with {
        Property InInterface : Interfaces = Client;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property SendsFirstMessage : SafeBoolean = Yes;
        Property InOurControlDomain : SafeBoolean = Yes;
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "requestAssistance";
            DT = "out" ], [
            ST = "assistanceOffers";
            DT = "in" ] >, < [
            ST = "requestAssistance";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

    Port confirmOffer : PortTWSClient =  new PortTWSClient extended with {
        Property InInterface : Interfaces = Client;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property SendsFirstMessage : SafeBoolean = Yes;
        Property InOurControlDomain : SafeBoolean = Yes;
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "confirmOffer";
```

```
                DT = "out" ], [
                ST = "offerConfirmation";
                DT = "in" ] >, < [
                ST = "confirmOffer";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port updateOffer : PortTWSClient =  new PortTWSClient extended with {
            Property InInterface : Interfaces = Client;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = Yes;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "updateOffer";
                DT = "out" ] >};
        };

        Port updateOffer2 : PortTWSClient =  new PortTWSClient extended with {
            Property InInterface : Interfaces = Client;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = Yes;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "updateOffer";
                DT = "out" ], [
                ST = "isUpdateAccepted";
                DT = "in" ] >, < [
                ST = "updateOffer";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port requestDiagnostic : PortTWSClient =  new PortTWSClient extended with {
            Property InInterface : Interfaces = Client;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = Yes;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "rawVehicleData";
                DT = "out" ], [
                ST = "diagnosticInformation";
                DT = "in" ] >, < [
                ST = "rawVehicleData";
                DT = "out" ], [
                ST = "fault";
```

```
                DT = "in" ] >};
        };

        Port requestDiagnostic2 : PortTWSClient =  new PortTWSClient extended with {
            Property InInterface : Interfaces = Client;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = Yes;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "rawVehicleDataAndChassisNumber";
                DT = "out" ], [
                ST = "diagnosticInformation";
                DT = "in" ] >, < [
                ST = "rawVehicleDataAndChassisNumber";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

    };

    Component NU : CompTWSClient =  new CompTWSClient extended with {
        Port getRoute : PortTWSClient =  new PortTWSClient extended with {
            Property InInterface : Interfaces = Client;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = Yes;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "routeCriteria";
                DT = "out" ], [
                ST = "pathData";
                DT = "in" ] >, < [
                ST = "routeCriteria";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port checkStatus : PortTWSClient =  new PortTWSClient extended with {
            Property InInterface : Interfaces = Client;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = No;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "requestStatusAndLocation";
                DT = "out" ], [
                ST = "statusAndLocation";
                DT = "in" ] >, < [
```

```
            ST = "requestStatusAndLocation";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

    Port updateRoute : PortTWSClient =  new PortTWSClient extended with {
        Property InInterface : Interfaces = Client;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property SendsFirstMessage : SafeBoolean = No;
        Property InOurControlDomain : SafeBoolean = Yes;
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "newPathData";
            DT = "out" ] >};
    };

    Port getEngineData : PortTWSClient =  new PortTWSClient extended with {
        Property InInterface : Interfaces = Client;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ], [
            Transport = HTTP1_0;
            Encoding = SOAP1_2 ]};
        Property SendsFirstMessage : SafeBoolean = Yes;
        Property InOurControlDomain : SafeBoolean = Yes;
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "requestData";
            DT = "out" ], [
            ST = "rawData";
            DT = "in" ] >, < [
            ST = "requestData";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

    Port requestAssistance : PortTWSClient =  new PortTWSClient extended with {
        Property InInterface : Interfaces = Client;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property SendsFirstMessage : SafeBoolean = Yes;
        Property InOurControlDomain : SafeBoolean = Yes;
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "requestAssistance";
            DT = "out" ], [
            ST = "assistanceOffers";
            DT = "in" ] >, < [
            ST = "requestAssistance";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
```

```
        };

        Port assistanceChoice : PortTWSClient =  new PortTWSClient extended with {
            Property InInterface : Interfaces = Client;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = Yes;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "assistanceChoice";
                DT = "out" ], [
                ST = "assistanceConfirmation";
                DT = "in" ] >, < [
                ST = "assistanceChoice";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port assistanceUpdate : PortTWSClient =  new PortTWSClient extended with {
            Property InInterface : Interfaces = Client;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = No;
            Property InOurControlDomain : SafeBoolean = Yes;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "updateOffer";
                DT = "out" ], [
                ST = "isUpdateAccepted";
                DT = "in" ] >, < [
                ST = "updateOffer";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

    };

    Component CM1E1 : CompTWSService =  new CompTWSService extended with {
        Port engineData : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property EndPointAddressList : EndPointAddresses = {"192.168.0.1/vehicleData"};
            Property SendsFirstMessage : SafeBoolean = No;
            Property InOurControlDomain : SafeBoolean = No;
            Property WsdlDocRefs : WsdlDocs = {"http://192.168.0.1/wsdl"};
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "requestData";
                DT = "out" ], [
                ST = "rawData";
```

```
            DT = "in" ] >, < [
            ST = "requestData";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

};

Component CM1E2 : CompTWSService =  new CompTWSService extended with {
    Port engineData : PortTWSService =  new PortTWSService extended with {
        Property InInterface : Interfaces = Service;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_2 ]};
        Property EndPointAddressList : EndPointAddresses = {"192.168.0.1/vehicleData"};
        Property SendsFirstMessage : SafeBoolean = No;
        Property InOurControlDomain : SafeBoolean = No;
        Property WsdlDocRefs : WsdlDocs = {"http://192.168.0.1/wsdl"};
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "requestData";
            DT = "out" ], [
            ST = "rawData";
            DT = "in" ] >, < [
            ST = "requestData";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

};

Component CM2E1 : CompTWSService =  new CompTWSService extended with {
    Port engineData : PortTWSService =  new PortTWSService extended with {
        Property InInterface : Interfaces = Service;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property EndPointAddressList : EndPointAddresses = {"192.168.0.1/vehicleData"};
        Property SendsFirstMessage : SafeBoolean = No;
        Property InOurControlDomain : SafeBoolean = No;
        Property WsdlDocRefs : WsdlDocs = {"http://192.168.0.1/wsdl"};
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "requestData";
            DT = "out" ], [
            ST = "rawData";
            DT = "in" ] >, < [
            ST = "requestData";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

};
```

```

    Component CM1 : CompTWSService =  new CompTWSService extended with {
        Port requestDiagnostic : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property EndPointAddressList : EndPointAddresses = {"cm1.com/getDiagnostic"};
            Property SendsFirstMessage : SafeBoolean = No;
            Property InOurControlDomain : SafeBoolean = No;
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.cm1.com"};
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "rawVehicleData";
                DT = "out" ], [
                ST = "diagnosticInformation";
                DT = "in" ] >, < [
                ST = "rawVehicleData";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

    };

    Component CM2 : CompTWSService =  new CompTWSService extended with {
        Port requestDiagnostic : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property EndPointAddressList : EndPointAddresses = {"cm2.com/getDiagnostic"};
            Property SendsFirstMessage : SafeBoolean = No;
            Property InOurControlDomain : SafeBoolean = No;
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.cm2.com"};
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "rawVehicleDataAndChassisNumber";
                DT = "out" ], [
                ST = "diagnosticInformation";
                DT = "in" ] >, < [
                ST = "rawVehicleDataAndChassisNumber";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

    };

    Component RS1 : CompTWSService =  new CompTWSService extended with {
        Port requestOffer : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = No;
```

```
            Property EndPointAddressList : EndPointAddresses = {"rs1.com/requestOffer"};
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.rs1.com"};
            Property InOurControlDomain : SafeBoolean = No;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "requestAssistance";
                DT = "out" ], [
                ST = "assistanceOffers";
                DT = "in" ] >, < [
                ST = "requestAssistance";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port confirmOffer : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = No;
            Property EndPointAddressList : EndPointAddresses = {"rs1.com/confirmOffer"};
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.rs1.com"};
            Property InOurControlDomain : SafeBoolean = No;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "confirmOffer";
                DT = "out" ], [
                ST = "offerConfirmation";
                DT = "in" ] >, < [
                ST = "confirmOffer";
                DT = "out" ], [
                ST = "fault";
                DT = "in" ] >};
        };

        Port updateOffer : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
                Transport = HTTP1_0;
                Encoding = SOAP1_1 ]};
            Property SendsFirstMessage : SafeBoolean = No;
            Property EndPointAddressList : EndPointAddresses = {"rs1.com/updateOffer"};
            Property WsdlDocRefs : WsdlDocs = {"http://wsdl.rs1.com"};
            Property InOurControlDomain : SafeBoolean = No;
            Property MessageExchangePatterns : messagePatterns = {< [
                ST = "updateOffer";
                DT = "out" ] >};
        };

    };

    Component RS2 : CompTWSService =  new CompTWSService extended with {
        Port requestOffer : PortTWSService =  new PortTWSService extended with {
            Property InInterface : Interfaces = Service;
            Property EndPointList : EndPoints = {[
```

```
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property SendsFirstMessage : SafeBoolean = No;
        Property EndPointAddressList : EndPointAddresses = {"rs2.com/requestOffer"};
        Property WsdlDocRefs : WsdlDocs = {"http://wsdl.rs2.com"};
        Property InOurControlDomain : SafeBoolean = No;
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "requestAssistance";
            DT = "out" ], [
            ST = "assistanceOffers";
            DT = "in" ] >, < [
            ST = "requestAssistance";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

    Port confirmOffer : PortTWSService =  new PortTWSService extended with {
        Property InInterface : Interfaces = Service;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property SendsFirstMessage : SafeBoolean = No;
        Property EndPointAddressList : EndPointAddresses = {"rs2.com/confirmOffer"};
        Property WsdlDocRefs : WsdlDocs = {"http://wsdl.rs2.com"};
        Property InOurControlDomain : SafeBoolean = No;
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "confirmOffer";
            DT = "out" ], [
            ST = "offerConfirmation";
            DT = "in" ] >, < [
            ST = "confirmOffer";
            DT = "out" ], [
            ST = "fault";
            DT = "in" ] >};
    };

    Port updateOffer : PortTWSService =  new PortTWSService extended with {
        Property InInterface : Interfaces = Service;
        Property EndPointList : EndPoints = {[
            Transport = HTTP1_0;
            Encoding = SOAP1_1 ]};
        Property SendsFirstMessage : SafeBoolean = No;
        Property EndPointAddressList : EndPointAddresses = {"rs2.com/updateOffer"};
        Property WsdlDocRefs : WsdlDocs = {"http://wsdl.rs2.com"};
        Property InOurControlDomain : SafeBoolean = No;
        Property MessageExchangePatterns : messagePatterns = {< [
            ST = "updateOffer";
            DT = "out" ], [
            ST = "isUpdateAccepted";
            DT = "in" ] >, < [
            ST = "updateOffer";
            DT = "out" ], [
            ST = "fault";
```

```
                DT = "in" ] >};
        };

    };

    Connector ConnTWS0 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment NU.getEngineData to ConnTWS0.r1;
    Attachment CM1E1.engineData to ConnTWS0.r2;
    Connector ConnTWS1 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment SNP.calcRoute to ConnTWS1.r1;
    Attachment NU.getRoute to ConnTWS1.r2;
    Connector ConnTWS2 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment SNP.updateRoute to ConnTWS2.r1;
    Attachment NU.updateRoute to ConnTWS2.r2;
    Connector ConnTWS3 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment SNP.requestAssistance to ConnTWS3.r1;
    Attachment NU.requestAssistance to ConnTWS3.r2;
    Connector ConnTWS4 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment SNP.assistanceChoice to ConnTWS4.r1;
    Attachment NU.assistanceChoice to ConnTWS4.r2;
    Connector ConnTWS5 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;
```

```

    };

    Attachment SNP.assistanceUpdate to ConnTWS5.r1;
    Attachment NU.assistanceUpdate to ConnTWS5.r2;
    Connector ConnTWS6 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment RS1.requestOffer to ConnTWS6.r1;
    Attachment SNP.requestOffer to ConnTWS6.r2;
    Connector ConnTWS7 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment RS1.confirmOffer to ConnTWS7.r1;
    Attachment SNP.confirmOffer to ConnTWS7.r2;
    Connector ConnTWS8 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment RS1.updateOffer to ConnTWS8.r1;
    Attachment SNP.updateOffer to ConnTWS8.r2;
    Connector ConnTWS9 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment CM1.requestDiagnostic to ConnTWS9.r1;
    Attachment SNP.requestDiagnostic to ConnTWS9.r2;
    Connector ConnTWS10 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment SNP.requestDiagnostic2 to ConnTWS10.r2;
    Attachment CM2.requestDiagnostic to ConnTWS10.r1;
    Connector ConnTWS11 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;
```

```
    };

    Attachment SNP.requestOffer to ConnTWS11.r2;
    Attachment RS2.requestOffer to ConnTWS11.r1;
    Connector ConnTWS12 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment SNP.confirmOffer to ConnTWS12.r2;
    Attachment RS2.confirmOffer to ConnTWS12.r1;
    Connector ConnTWS13 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment SNP.updateOffer2 to ConnTWS13.r1;
    Attachment RS2.updateOffer to ConnTWS13.r2;
    Connector ConnTWS14 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment CM1E2.engineData to ConnTWS14.r2;
    Attachment NU.getEngineData to ConnTWS14.r1;
    Connector ConnTWS15 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment CM2E1.engineData to ConnTWS15.r2;
    Attachment NU.getEngineData to ConnTWS15.r1;
    Connector ConnTWS16 : ConnTWS =  new ConnTWS extended with {
        Role r1;

        Role r2;

    };

    Attachment NU.checkStatus to ConnTWS16.r2;
    Attachment SNP.checkStatus to ConnTWS16.r1;
};
```

Appendix D

Enhanced Style Description

D.1 Rules for using the style

The style and analysis makes three assumptions about the CSP properties within a system, all of which are syntactic. These are:

D.1.1 Port message pattern naming

The analysis requires that the process IDs in each port's `messagePattern` property are unique within a system. A suggested structure to ensure this is to name each process with the qualified name of the port it exists within. For example the message pattern process of port 'port1' on component 'comp1' would be 'comp1-port1'. This naming structure should also be included in the following lines of message pattern template. An example of this from a port named `setupConf` on the component `CPClient` can be seen in Figure D.1.

D.1.2 Message naming

The analysis also requires that the names given to each message in the message pattern CSP descriptions are unique within the system. The suggested structure here is an extension of that suggested for the ports, i.e. the qualified name of the port followed by the message name within the port. For example a 'login' message in the above port would be named 'comp1-p1-login'. This naming

```
Property MessagePattern = "SOLI
  CPClient_setupConf = CPClient_setupConf_sendReq -> CPClient_setupConf_p1
  CPClient_setupConf_p1 = CPClient_setupConf_p2 [] CPClient_setupConf_p3
  CPClient_setupConf_p2 = CPClient_setupConf_getRes -> CPClient_setupConf_OK
  CPClient_setupConf_p3 = CPClient_setupConf_getFault -> CPClient_setupConf_FAULT
  CPClient_setupConf_OK = CPClient_PaymentCC
  CPClient_setupConf_FAULT = CPClient_PaymentCC";
```

Figure D.1: An example `messagePattern` property from a port in the car parking scenario listed in Appendix E.

structure can be seen employed in Figure D.1.

D.1.3 Forbidden message name

The naming structures are suggested but are not mandatory. In the case that they are not followed there is a single message ID that should be avoided. This is *faux*. This name is used to represent a message that will not exist when stubborn connectors exist in a system. Using it as a message name could lead to false results being returned by all the analyses based upon the CSP model of the system.

D.2 The Style Definition

```
Family ws_enhanced_01 = {

    // Below are the declarations of the external analyses used in the style.  The declaration
    // takes follow the form "external analysis <rulename><formal parameters> : <return type>
    // = <java class and path>;".  The external analysis

    external analysis EAMessageExchangePatternsMatch(thisConnector : Element) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.MessageExchangePatternsMatch;

    external analysis EAMessageExchangePatternsPartiallyMatch(thisConnector : Element) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.MessageExchangePatternsPartiallyMatch;

    external analysis EAConcurrentCallsToThisPort(thisPort : Element) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.ConcurrentCallsToThisPort;

    external analysis EACentralDataStoreCorrect(thisComponent : Element) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.CentralDataStoreCorrect;

    external analysis EACommissionMismatch(thisComponent : Element) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.CommissionMismatch;

    external analysis EACommissionPartialMatch(thisComponent : Element) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.CommissionPartialMatch;

    external analysis EAOmissionMismatch(thisComponent : Element) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.OmissionMismatch;

    external analysis EAOmissionPartialMatch(thisComponent : Element) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.OmissionPartialMatch;

    external analysis EAMessageDataTypesMatch(thisConnector : Element,firstPort : Element
        ,secondPort : Element,messageNo : int) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.MessageDataTypesMatch;

    external analysis EAMessageOverData(thisConnector : Element,firstPort : Element
        ,secondPort : Element,messageNo : int) : boolean
        = uk.ac.ncl.cjg.ws_enhanced.MessageOverData;
```

```

external analysis EAMessageUnderData1(thisConnector : Element,firstPort : Element
    ,secondPort : Element,messageNo : int) : boolean
    = uk.ac.ncl.cjg.ws_enhanced.MessageUnderData1;

external analysis EAMessageUnderData2(thisConnector : Element,firstPort : Element
    ,secondPort : Element,messageNo : int) : boolean
    = uk.ac.ncl.cjg.ws_enhanced.MessageUnderData2;

external analysis EAStateScopesMatch(thisConnector : Element,firstPort : Element
    ,secondPort : Element) : boolean
    = uk.ac.ncl.cjg.ws_enhanced.StateScopesMatch;

external analysis EAMessagePatternAndMessageListConcur(thisPort : Element) : boolean
    = uk.ac.ncl.cjg.ws_enhanced.MessagePatternAndMessageListConcur;

external analysis EAChoiceGroupsHaveChoiceMaker(thisComponent : Element) : boolean
    = uk.ac.ncl.cjg.ws_enhanced.ChoiceGroupsHaveChoiceMaker;

// Below are the custom types used in this style, the syntax does not allow them
// to be defined in the connectors where the properties based upon them are
// instantiated



// The following types support the definition of the messages exchange
// and the data they contain

Property Type TMessage = Record [
     MessageId : string;
     MessageData : Set {TMessageDatum};
];

Property Type TMessages = set{TMessage};

Property Type TMessageDatum = Record [
     DatumId : string;
     DatumRep : TDataRep;
     DatumStateScopeExpected : TStateScopeExpected;
];

Property Type TDataRep = Enum {
     SOAP_Int,
     SOAP_String,
     SOAP_Float,
     SOAP_Bool,
     SOAP_Date,
     SOAP_Time,
     SOAP_DateTime
};

Property Type TCentralDataRecord = Record [
     DatumID : string;
     DatumSemantics : TDataSemantics;
```

```
          DatumScopeExhibited : TStateScopeExhibited;
     ];

     Property Type TDataSemantics = string;


     // Two types supporting the scope over which an element of data
     // is expected to be shared and the maximum scope over which a component
     // states it may share it.

     Property Type TStateScopeExhibited = Enum {
          Private,
          Shared
     };

     Property Type TStateScopeExpected = Enum {
          Private,
          Shared,
          NoPreference
     };


     // These types support the definition of an adressable endpoint in terms of
     // their transport encoding protocols and address.  The address is only applicable
     // to service type ports that are required to be discoverable.

     Property Type TLegalSoapVersions = Enum {
          SOAP1_1,
          SOAP1_2
     };

     Property Type TLegalTransportProtocols = Enum {
          HTTP1_0,
          HTTP1_1
     };

     Property Type TEndPoint = Record [
          Transport : TLegalTransportProtocols;
          Encoding : TLegalSoapVersions;
     ];

     Property Type TEndPointAddresses = Set {string};

     Property Type TEndPoints = Set {TEndPoint};


     // Types used to indicate types of failure a port might exhibit or
     // that a port may assume another port may exhibit and therefore contain
     // handlers for.

     Property Type TFailureMode = Enum {
          ContentFailures,
          EarlyTimingFailures,
          LateTimingFailures,
```

```
          HaltFailures,
          ErraticFailures
     };

     Property Type TFailureModes = Set {ws_enhanced_01.TFailureMode};


     // An enumeration of the allowed binding times in the style

     Property Type TBindTime = Enum {Write,Compile,Instantiation,Run};


     // A property type used to allow ACME Studio to distinguish between
     // client and service ports correctly

     Property Type TInterfaces = Enum {
          Client,
          Service
     };


     // The simple type used to contain the CSP descriptions in the
     // system.

     Property Type TCSP = string;


     // A type to hold the addresses of the WSDL documents referring to a specific port

     Property Type TWsdlDocs = Set {string};


     // A type to indicate the continuity of data availability either expected or
     // exhibited by a port.

     Property Type TDataContinuity = Enum {
          Sporadic,
          Continuous
     };


     // A work-a-round alternative for the built in boolean for which there is no means
     // to positively identify a property that has not been initialised

     Property Type TSafeBoolean = Enum {
          Yes,
          No
     };

     // A type to allow a port to have no preference whether the other port can create
     // or destroy a particular connection

     Property Type TConnCreationDestructionAssumption = Enum {
          May,
```

```
        MayNot,
        Either
    };

    // Below are the component types created in the style.
    // The component heirarchy is :
    //                CompTWSCommom                              CompTWSAnalysisControl
    // CompTWSClient   CompTWSService  CompTWSIntermediary


    Component Type CompTWSCommon = {

        Property CentralProcessDescription : TCSP;

        Property CentralDataRecords : Set {TCentralDataRecord};

        Property ComponentInOurControlDomain : TSafeBoolean;

        rule CentralProcessDescribed = invariant CentralProcessDescription != ""
            <<  label : string = "Components Central CSP process Description has contents";
            errMsg : string = "The Central CSP process description is empty"; >>;

        rule ComponentInOurControlDomainDescribed = invariant ComponentInOurControlDomain == Yes
                                                            OR ComponentInOurControlDomain == No
            <<  label : string = "The ComponentInOurControlDomain property is populated";
            errMsg : string = "The ComponentInOurControlDomain property is not populated"; >>;

        rule MsgDatumDescribed = invariant EACentralDataStoreCorrect(self)
            <<  label : string = "All data in the messages is represented in the central
                                 data store";
            errMsg : string = "Data represented in a message does not exist in central data
                              store, check the analysis output for details"; >>;

        rule ChoiceGroupsHaveChoiceMakers = invariant EAChoiceGroupsHaveChoiceMaker(self)
            <<  label : string = "All choice groups in this component have their own choice
                                 makers";
            errMsg : string = "One or more choice groups are missing a choice maker, check the
                              analysis output for details"; >>;

        rule CommissionMismatch = invariant EACommissionMismatch(self)
            <<  label : string = "This component does not send any unexpected messages to its
                                 environment - where neither port is in our control";
            errMsg : string = "The component sends one or more unexpected messages to the
                              environment, neither port is in our control, see analysis
                              output for details."; >>;

        rule CommissionPartialMatch = invariant EACommissionPartialMatch(self)
            <<  label : string = "This component does not send any unexpected messages to its
                                 environment - where one or bort ports is in our control";
            errMsg : string = "The component sends one or more unexpected messages to the
                              environment where one or both ports involved is in our control,
                              see analysis output for details."; >>;

        rule OmissionMismatch = invariant EAOmissionMismatch(self)
```

```
            <<  label : string = "This component receives all expected messages on connections
                                  where neither port is in our control";
            errMsg : string = "The port does not receive one or more expected messages on
                               connections where neither port is in our control"; >>;

        rule OmissionPartialMatch = invariant EAOmissionPartialMatch(self)
            <<  label : string = "This component receives all expected messages on connections
                                  where one or both ports are in our control";
            errMsg : string = "This component does not receive one or more messages on
                               connections where one or both ports are in our control,
                               see analysis output for details"; >>;
    }


    Component Type CompTWSClient extends CompTWSCommon with {

        rule AllClientPorts = invariant forall p : Port in self.PORTS |
            satisfiesType(p, PortTWSClientSingle)
            OR satisfiesType(p, PortTWSClientUnicast)
                <<  label : string = "External ports are all Client type";
                errMsg : string = "Only client type ports are allowed"; >> ;

        rule ComponentHasValidPorts = invariant size(self.PORTS) > 0
            <<  label : string = "Component has at least one port";
            errMsg : string = "Component should have at least one port"; >> ;

    }


    Component Type CompTWSIntermediary extends CompTWSCommon with {

        rule ComponenthasValidPorts = invariant forall p : Port in self.PORTS |
            satisfiesType(p, PortTWSClientSingle)
            OR satisfiesType(p, PortTWSClientUnicast)
            OR satisfiesType(p, PortTWSServiceSingle)
            OR satisfiesType(p, PortTWSServiceUnicast)
            <<  label : string = "External ports are of the web service type";
            errMsg : string = "Only WebService type ports are allowed"; >> ;

        rule ComponentHasClientInterface = invariant exists p : Port in self.PORTS |
            satisfiesType(p, PortTWSClientSingle)
            OR satisfiesType(p, PortTWSClientUnicast)
            <<  label : string = "Component has at least one client type port";
            errMsg : string = "Component must have at least one client type port"; >> ;

        rule ComponentHasServiceInterface = invariant exists p : Port in self.PORTS |
            satisfiesType(p, PortTWSServiceSingle)
            OR satisfiesType(p, PortTWSServiceUnicast)
            <<  label : string = "Component has at least one service type port";
            errMsg : string = "Component must have at least one service type port"; >> ;
    }


    Component Type CompTWSService extends CompTWSCommon with {
```

```
        rule AllServicePorts = invariant forall p : Port in self.PORTS |
            satisfiesType(p, PortTWSServiceSingle)
            OR satisfiesType(p, PortTWSServiceUnicast)
            <<  label : string = "External ports are all Service type";
            errMsg : string = "Only service type ports are allowed"; >> ;

        rule ComponentHasValidPorts = invariant size(self.PORTS) > 0
             <<  label : string = "Component has at least one port";
             errMsg : string = "Component should have at least one port"; >> ;
    }


    Component Type CompTWSAnalysisControl = {

        Property ActiveAnalysisCommissionMismatch : boolean;

        Property ActiveAnalysisCommissionPartialMatch : boolean;

        Property ActiveAnalysisOmissionMismatch : boolean;

        Property ActiveAnalysisOmissionPartialMatch : boolean;

        Property ActiveAnalysisMessageExchangePatternsMatch : boolean;

        Property ActiveAnalysisMessageExchangePatternsPartiallyMatch : boolean;

        Property ActiveAnalysisConcurrentCallsToThisPort : boolean;

        Property ActiveAnalysisCentralDataStoreCorrect : boolean;

        Property ActiveAnalysisMessageDataTypesMatch : boolean;

        Property ActiveAnalysisMessageOverData : boolean;

        Property ActiveAnalysisMessageUnderData1 : boolean;

        Property ActiveAnalysisMessageUnderData2 : boolean;

        Property ActiveAnalysisStateScopesMatch : boolean;

        Property ActiveAnalysisMessagePatternAndMessageListConcur : boolean;

        Property ActiveAnalysisChoiceGroupsHaveChoiceMaker : boolean;

        Property outputPath : string;

        rule AnalysisCommissionMismatchActive
            = invariant ActiveAnalysisCommissionMismatch
            <<  label : string = "Message commission mismatch : analysis active";
            errMsg : string = "Message commission mismatch : analysis inactive"; >>;

        rule AnalysisCommissionPartialMatchActive
            = invariant ActiveAnalysisCommissionPartialMatch
            <<  label : string = "Message commission partial mismatch : analysis active";
```

```
            errMsg : string = "Message commission partial mismatch : analysis inactive"; >>;

        rule AnalysisOmissionMismatchActive
            = invariant ActiveAnalysisOmissionMismatch
            <<  label : string = "Message omission mismatch : analysis active";
            errMsg : string = "Message omission mismatch : analysis inactive"; >>;

        rule AnalysisOmissionPartialMatchActive
            = invariant ActiveAnalysisOmissionPartialMatch
            <<  label : string = "Message omission partial mismatch : analysis active";
            errMsg : string = "Message omission partial mismatch : analysis inactive"; >>;

        rule AnalysisMessageExchangePatternsMatchActive
            = invariant ActiveAnalysisMessageExchangePatternsMatch
            <<  label : string = "Message exchange pattern match : analysis active";
            errMsg : string = "Message exchange pattern match : analysis inactive"; >>;

        rule AnalysisMessageExchangePatternsPartiallyMatchActive
            = invariant ActiveAnalysisMessageExchangePatternsPartiallyMatch
            <<  label : string = "Message exchange pattern partial match : analysis active";
            errMsg : string = "Message exchange pattern partial match : analysis inactive"; >>;

        rule AnalysisConcurrentCallsToThisPortActive
            = invariant ActiveAnalysisConcurrentCallsToThisPort
            <<  label : string = "Concurrent calls to a non reentrant port : analysis active";
            errMsg : string = "Concurrent calls to a non reentran port : analysis inactive"; >>;

        rule AnalysisCentralDataStoreCorrectActive
            = invariant ActiveAnalysisCentralDataStoreCorrect
            <<  label : string = "Confirmation that message data is represented in central data
                                  store : analysis active";
            errMsg : string = "Confirmation that message data is represented in central data
                               store : analysis inactive"; >>;

        rule AnalysisMessageDataTypesMatchActive
            = invariant ActiveAnalysisMessageDataTypesMatch
            <<  label : string = "Data types match in messages exchanged : analysis active";
            errMsg : string = "Data types match in messages exchanged : analysis inactive"; >>;

        rule AnalysisMessageOverDataActive
            = invariant ActiveAnalysisMessageOverData
            <<  label : string = "Message contains unrequired data : analysis active";
            errMsg : string = "Message contains unrequired data : analysis inactive"; >>;

        rule AnalysisMessageUnderData1Active
            = invariant ActiveAnalysisMessageUnderData1
            <<  label : string = "Message does not contain required data : analysis active";
            errMsg : string = "Message does not contain required data : analysis inactive"; >>;

        rule AnalysisMessageUnderData2Active
            = invariant ActiveAnalysisMessageUnderData2
            <<  label : string = "Message does not contain required data : analysis active";
            errMsg : string = "Message does not contains required data : analysis inactive"; >>;
```

```
        rule AnalysisStateScopesMatchActive
            = invariant ActiveAnalysisStateScopesMatch
            <<  label : string = "Expected and exhibited state scopes : analysis active";
            errMsg : string = "Expected and exhibited state scopes : analysis inactive"; >>;

        rule AnalysisMessagePatternAndMessageListConcurActive
            = invariant ActiveAnalysisMessagePatternAndMessageListConcur
            <<  label : string = "Message names in port CSP and messages property match :
                                  analysis active";
            errMsg : string = "Message names in port CSP and messages property match :
                               analysis inactive"; >>;

        rule AnalysisChoiceGroupsHaveChoiceMakerActive
            = invariant ActiveAnalysisChoiceGroupsHaveChoiceMaker
            <<  label : string = "Confirmation that choice groups have a choice maker :
                                  analysis active";
            errMsg : string = "Confirmation that choice groups have a choice maker :
                               analysis inactive"; >>;
    }


    // Below are the port types created in the style.
    // Their heirarchy is as follows :
    //                                        PortTWSCommon
    //                PortTWSClient                                    PortTWSService
    //  PortTWSClientUnicast PortTWSClientSingle    PortTWSServiceSingle PortTWSServiceUnicast



    Port Type PortTWSCommon = {

        Property EndPointList : TEndPoints;

        Property InOurControlDomain : TSafeBoolean;

        Property SendsFirstMessage : TSafeBoolean;

        Property FailureModesExpected : TFailureModes;

        Property FailureModesExhibited : TFailureModes;

        Property Reentrant : TSafeBoolean;

        Property Messages : TMessages;

        Property BindTime : TBindTime;

        Property BindingSelfAdd : TConnCreationDestructionAssumption;

        Property BindingSelfRemove : TConnCreationDestructionAssumption;

        Property BindingOtherAdd : TConnCreationDestructionAssumption;

        Property BindingOtherRemove : TConnCreationDestructionAssumption;
```

```
        Property MessagePattern : TCSP;

        Property DataContinuity : TDataContinuity;

        rule EndpointListPopulated = invariant size(EndPointList) > 0
             << label : string = "Endpoint list is populated";
             errMsg : string = "Endpoint list must be populated"; >> ;

        rule InOurControlDomainPopulated = invariant InOurControlDomain == Yes
            OR InOurControlDomain == No
            << label : string = "In our control domain property is populated";
            errMsg : string = "In Our Control Domain property must be populated"; >> ;

        rule SendsFirstMessagePopulated = invariant SendsFirstMessage == Yes
            OR SendsFirstMessage == No
            << label : string = "Sends first message property is populated";
            errMsg : string = "Sends First Message property must be populated"; >> ;

        rule PortReentered = invariant Reentrant == Yes
            OR EAConcurrentCallsToThisPort(self) == true
            << label : string = "No reentrance problems with this port";
            errMsg : string = "Reentrance problem detected with this port, see analysis
                               output for details"; >>;

        rule MsgNamesConsistent = invariant EAMessagePatternAndMessageListConcur(self)
            << label : string = "All messages in the CSP pattern are included in the
                                  messages property";
            errMsg : string = "One or more messages in the CSP patter is not included in
                               the message property, see analysis output for details."; >>;

        rule ReentrantPopulated = invariant Reentrant == Yes
            OR Reentrant == No
            << label : string = "Port reentrance property is populated";
            errMsg : string = "Port reentrance property is not populated"; >>;

        rule BindingSelfAddPopulated = invariant BindingSelfAdd == May
            OR BindingSelfAdd == MayNot
            << label : string = "BindingSelfAdd property populated";
            errMsg : string = "BindingSelfAdd property is not populated or may
                               be set to Either which is not allowed"; >> ;

        rule BindingSelfRemovePopulated = invariant BindingSelfRemove == May
            OR BindingSelfRemove == MayNot
            << label : string = "BindingSelfRemove property populated";
            errMsg : string = "BindingSelfRemove property is not populated or may
            be set to Either which is not allowed"; >> ;

        rule BindingOtherAddPopulated = invariant BindingOtherAdd == May
            OR BindingOtherAdd == MayNot OR BindingOtherAdd == Either
            << label : string = "BindingOtherAdd property populated";
            errMsg : string = "BindingOtherAdd property is not populated"; >> ;

        rule BindingOtherRemovePopulated = invariant BindingOtherRemove == May
```

```
            OR BindingOtherRemove == MayNot OR BindingOtherRemove == Either
            <<  label : string = "BindingOtherRemove property populated";
            errMsg : string = "BindingOtherRemove property is not populated"; >> ;

        rule MessagePatternPopulated = invariant MessagePattern != ""
            <<  label : string = "Port CSP message pattern property is not empty";
            errMsg : string = "Port CSP pattern property is empty"; >>;


        rule DataContinuityPopulated = invariant DataContinuity == Sporadic
            OR DataContinuity == Continuous
            <<  label : string = "Data Continuity property populated";
            errMsg : string = "Data continuity property is not populated"; >>;
    }


    Port Type PortTWSClient extends PortTWSCommon with {

        Property InInterface : TInterfaces = Client;

        rule BindingTimePopulated = invariant BindTime == Write
            OR BindTime == Compile
            OR BindTime == Instantiation
            OR BindTime == Run
            <<  label : string = "Port binding time is populated";
            errMsg : string = "port binding time is not populated"; >>;
    }


    Port Type PortTWSClientSingle extends PortTWSClient with {

        rule CardinalityOfAttachmentsOK = invariant size(self.ATTACHEDROLES) == 1
            <<  label : string = "Port is attached to an acceptable number of connectors";
            errMsg : string = "Port is attached to too many or too few connectors"; >>;
    }


    Port Type PortTWSClientUnicast extends PortTWSClient with {

        Property ChoiceGroup : string;

        Property GroupChoiceMaker : TSafeBoolean;

        rule ChoiceGroupPopulated = invariant ChoiceGroup != ""
            <<  label : string = "Choice group is populated";
            errMsg : string = "Choice group property is empty"; >>;

        rule CardinalityOfAttachmentsOK = invariant size(self.ATTACHEDROLES) > 0
            <<  label : string = "Port is attached to an acceptable number of connectors";
            errMsg : string = "Port is attached to too few connectors"; >>;
    }


    Port Type PortTWSService extends PortTWSCommon with {
```

```

        Property InInterface : TInterfaces = Service;

        Property EndPointAddressList : TEndPointAddresses;

        Property WsdlDocRefs : TWsdlDocs;

        rule EndPointAddressPopulated = invariant size(EndPointAddressList) > 0
            <<  label : string = "Endpoint address list is populated";
            errMsg : string = "Endpoint address list must be populated"; >> ;

        rule EachEndpointProtocolAddressed =
            invariant size(EndPointAddressList)== size(EndPointList)
            <<  label : string = "Number EndPoint addresses = number of EndPoint protocol pairs";
            errMsg : string = "Must be one End Point Address for each End Point protocol pair";
                            >> ;

        rule HasWSDL = invariant size(WsdlDocRefs) > 0
            <<  label : string = "WSDL reference list is populated";
            errMsg : string = "WSDL reference list should be populated"; >> ;

        rule StatedBindingTime = invariant BindTime == Instantiation
            OR BindTime == Run
            <<  label : string = "Binding time is populated correctly";
            errMsg : string = "Binding time is either empty or has a disallowed value"; >>;
    }


    Port Type PortTWSServiceSingle extends PortTWSService with {

        rule CardinalityOfAttachmentsOK = invariant size(self.ATTACHEDROLES) == 1
            <<  label : string = "Port is attached to an acceptable number of connectors";
            errMsg : string = "Port is attached to too many or too few connectors"; >>;
    }


    Port Type PortTWSServiceUnicast extends PortTWSService with {

        Property ChoiceGroup : string;

        Property GroupChoiceMaker : TSafeBoolean;

        rule ChoiceGroupPopulated = invariant ChoiceGroup != ""
            <<  label : string = "Choice group is populated";
            errMsg : string = "Choice group property is empty"; >>;

        rule CardinalityOfAttachmentsOK = invariant size(self.ATTACHEDROLES) > 0
            <<  label : string = "Port is attached to an acceptable number of connectors";
            errMsg : string = "Port is attached to too few connectors"; >>;
    }


    // Below are the connector types created in the style.
    // There is no heirarchy as the two types are completely independant with ConnTWS being
```

```
    // used to represent all known connections in the system and the ConnTWSCooperative
    // representing links to unknown parts of the system.

    Connector Type ConnTWS = {

        Role role1  = {
        }

        Role role2  = {
        }

        rule CorrectNumberOfRoles = invariant size(self.ROLES) == 2
            <<  label : string = "A connector of this type must have 2 roles";
            errMsg : string = "This connector must have exactly two roles"; >> ;

        rule EndpointProtocols = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> size(intersection(p1.EndPointList, p2.EndPointList)) > 0
            <<  label : string = "Ports have a matching Transport / Encoding pair";
            errMsg : string = "No matching pair of endpoint protocols"; >> ;

        rule OnePortSendsFirstMessage = invariant exists r : Role in self.ROLES |
            forall p : PortTWSCommon in r.ATTACHEDPORTS |
            attached(r, p)
            -> p.SendsFirstMessage == Yes
            <<  label : string = "One port expects to send the first message";
            errMsg : string = "Neither port expects to send the first message"; >> ;

        rule OnePortReceivesFirstMessage = invariant exists r : Role in self.ROLES |
            forall p : PortTWSCommon in r.ATTACHEDPORTS |
            attached(r, p)
            -> p.SendsFirstMessage == No
            <<  label : string = "One port is listening for the first message";
            errMsg : string = "Neither port is listening for the first message"; >> ;

        rule MessageExchangePatternsMatch = invariant EAMessageExchangePatternsMatch(self)
            <<  label : string = "The message exchange patterns match or there may be a partial
                                  match, check the other rule";
            errMsg : string = "The message exchange patterns do not match and neither port is in
                               our control, see analysis output for details."; >>;

        rule MessageExchangePatternsPartiallyMatch = invariant
             EAMessageExchangePatternsPartiallyMatch(self)
            <<  label : string = "The message exchange pattern either matches completely or there
                                  is a mismatch, check the other rule.";
            errMsg : string = "There is a partial match between the message exchange patterns,
                               see the analysis output for details."; >>;


        rule MatchingDataContinuityAssumptions = invariant forall p1 : PortTWSCommon in role1.
             ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
```

```
            attached(role1, p1)
            AND attached(role2, p2)
            -> p1.DataContinuity == p2.DataContinuity
            <<  label : string = "The data continuity assumptions of both ports match";
            errMsg : string = "The data continuity assumptions do not match"; >>;


        rule Msg1MessageDataTypesMatch = invariant forall p1 : PortTWSCommon in role1.
            ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> EAMessageDataTypesMatch(self, p1, p2, 1)
            <<  label : string = "The message data types in the first message in the
                                  pattern match";
            errMsg : string = "There is a mismatch in the data exchanged in the first message,
                               see the analysis output for details."; >>;


        rule Msg1MessageOverData = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> EAMessageOverData(self, p1, p2, 1)
            <<  label : string = "There is no redundant information in the first message sent";
            errMsg : string = "The first message sent contains information not required by the
                               recipient, see the analysis output for details."; >>;

        rule Msg1MessageUnderData1 = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> EAMessageUnderData1(self, p1, p2, 1)
            <<  label : string = "There is no data missing from the first message that is
                                  required by the recipient that the sender may be able to send";
            errMsg : string = "There is data missing from the first message that the recipient
                               requires that the sender may be able to send, see the analysis
                               output for details"; >>;

        rule Msg1MessageUnderData2 = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> EAMessageUnderData2(self, p1, p2, 1)
            <<  label : string = "There is no data missing from the first message that is
                                  required by the recipient that the sender is unable to send";
            errMsg : string = "There is data missing from the first message that the recipient
                               requires that the sender is unable to send, see the analysis
                               output for details"; >>;

        rule Msg2MessageDataTypesMatch = invariant forall p1 : PortTWSCommon in role1.
            ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
```

```
        AND attached(role2, p2)
        -> EAMessageDataTypesMatch(self, p1, p2, 2)
        <<  label : string = "The message data types in the second message in the
                              pattern match";
        errMsg : string = "There is a mismatch in the data exchanged in the second message,
                           see the analysis output for details."; >>;

    rule Msg2MessageOverData = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageOverData(self, p1, p2, 2)
        <<  label : string = "There is no redundant information in the second message sent";
        errMsg : string = "The second message sent contains information not required by the
                           recipient, see the analysis output for details."; >>;


    rule Msg2MessageUnderData1 = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageUnderData1(self, p1, p2, 2)
        <<  label : string = "There is no data missing from the second message that is
                              required by the recipient that the sender may be able to send";
        errMsg : string = "There is data missing from the second message that the recipient
                           requires that the sender may be able to send, see the analysis
                           output for details"; >>;

    rule Msg2MessageUnderData2 = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageUnderData2(self, p1, p2, 2)
        <<  label : string = "There is no data missing from the second message that is
                           required by the recipient that the sender is unable to send";
        errMsg : string = "There is data missing from the second message that the recipient
                           requires that the sender is unable to send, see the analysis
                           output for details"; >>;

    rule Msg3MessageDataTypesMatch = invariant forall p1 : PortTWSCommon in role1.
        ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageDataTypesMatch(self, p1, p2, 3)
        <<  label : string = "The message data types in the third message in the
                              pattern match";
        errMsg : string = "There is a mismatch in the data exchanged in the third message,
                           see the analysis output for details."; >>;

    rule Msg3MessageOverData = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
```

```
        -> EAMessageOverData(self, p1, p2, 3)
        <<  label : string = "There is no redundant information in the third message sent";
        errMsg : string = "The third message sent contains information not required by the
                           recipient, see the analysis output for details."; >>;

    rule Msg3MessageUnderData1 = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageUnderData1(self, p1, p2, 3)
        <<  label : string = "There is no data missing from the third message that is
                           required by the recipient that the sender may be able to send";
        errMsg : string = "There is data missing from the third message that the recipient
                           requires that the sender may be able to send, see the analysis
                           output for details"; >>;

    rule Msg3MessageUnderData2 = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageUnderData2(self, p1, p2, 3)
        <<  label : string = "There is no data missing from the third message that is
                             required by the recipient that the sender is unable to send";
        errMsg : string = "There is data missing from the third message that the recipient
                         requires that the sender is unable to send, see the analysis output
                         for details"; >>;

    rule Msg4MessageDataTypesMatch = invariant forall p1 : PortTWSCommon in role1.
         ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageDataTypesMatch(self, p1, p2, 4)
        <<  label : string = "The message data types in the fourth message in the
                              pattern match";
        errMsg : string = "There is a mismatch in the data exchanged in the fourth message,
                           see the analysis output for details."; >>;

    rule Msg4MessageOverData = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageOverData(self, p1, p2, 4)
        <<  label : string = "There is no redundant information in the fourth message sent";
        errMsg : string = "The first message sent contains information not required by the
                           recipient, see the analysis output for details."; >>;

    rule Msg4MessageUnderData1 = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
        forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
        attached(role1, p1)
        AND attached(role2, p2)
        -> EAMessageUnderData1(self, p1, p2, 4)
        <<  label : string = "There is no data missing from the fourth message that is
                             required by the recipient that the sender may be able to send";
```

```
            errMsg : string = "There is data missing from the fourth message that the recipient
                            requires that the sender may be able to send, see the analysis output
                            for details"; >>;

        rule Msg4MessageUnderData2 = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> EAMessageUnderData2(self, p1, p2, 4)
            <<  label : string = "There is no data missing from the fourth message that is
                                  required by the recipient that the sender is unable to send";
            errMsg : string = "There is data missing from the fourth message that the recipient
                          requires that the sender is unable to send, see the analysis output
                          for details"; >>;

        rule StateScopeAssumptionsMatch = invariant forall p1 : PortTWSCommon in role1.
             ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> EAStateScopesMatch(self, p1, p2)
            <<  label : string = "The state scope assumptions of both ports match";
            errMsg : string = "There is a mismatch in the state scope assumptions, see
                               the analysis output for details"; >>;

       rule ConnectorCreationDestruction = invariant forall p1 : PortTWSCommon in role1.
            ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1) AND attached(role2, p2)
            -> (p1.BindingOtherAdd == p2.BindingSelfAdd
                OR(p1.BindingOtherAdd == Either AND p1.BindingSelfAdd == May))
              AND
               (p1.BindingOtherRemove == p2.BindingSelfRemove
                OR(p1.BindingOtherRemove == Either AND p1.BindingSelfRemove == May))
              AND
               (p2.BindingOtherAdd == p1.BindingSelfAdd
                OR(p2.BindingOtherAdd == Either AND p2.BindingSelfAdd == May))
              AND
               (p2.BindingOtherRemove == p1.BindingSelfRemove
                OR(p2.BindingOtherRemove == Either AND p2.BindingSelfRemove == May))
              <<  label : string = "The connector creation and destruction assumed
                                    permissions are compatible";
                  errMsg : string = "There is a mismatch in the connector creation
                                    and destruction assumed permissions."; >>;

      rule SaneConnectorCreationDestruction = invariant forall p1 : PortTWSCommon in role1.
           ATTACHEDPORTS |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> (p1.BindingSelfAdd == May OR p2.BindingSelfAdd == May)
               AND
               (p1.BindingSelfRemove == May OR p2.BindingSelfRemove == May)
                <<  label : string = "The assumed permissions for connector creation and
```

```
                                      destruction are realistic";
                errrMsg : string = "The assumed permissions for connector creation and destruction
                            do not allow the connector to be either created or destoyed."; >>;

        rule FailureModeAssumptions = invariant forall p1 : PortTWSCommon in role1.ATTACHEDPORTS
            |
            forall p2 : PortTWSCommon in role2.ATTACHEDPORTS |
            attached(role1, p1)
            AND attached(role2, p2)
            -> (isSubset(p1.FailureModesExhibited, p2.FailureModesExpected))
               AND
               (isSubset(p2.FailureModesExhibited, p1.FailureModesExpected))
               <<  label : string = "The failure mode expected cover all those exhibited";
               errMsg : string = "There are failure modes exhibited that are not expected by the
                                 other port."; >>;
    }


    Connector Type ConnTWSCooperative = {
        Role role1  = {
        }
    }


    Connector Type ConnTWSStubborn = {
            Role role1  = {
        }
    }


    rule NatureOfComponents = invariant forall comp : Component in self.COMPONENTS |
        satisfiesType(comp, CompTWSClient)
        OR satisfiesType(comp, CompTWSService)
        OR satisfiesType(comp, CompTWSIntermediary)
        OR satisfiesType(comp, CompTWSAnalysisControl)
        <<  label : string = "All components are WSClients, WSServices WSIntermediary
                              or WSAnalysisControl";
        errMsg : string = "Style only permits WSClient, WSService, WSIntermediary and
                           WSAnalysisControl type components"; >> ;

    rule NatureOfConnectors = invariant forall conn : Connector in self.CONNECTORS |
        satisfiesType(conn, ConnTWS)
        OR satisfiesType(conn, ConnTWSCooperative)
        OR satisfiesType(conn, ConnTWSStubborn)
        <<  label : string = "All Connectors are WS type";
        errMsg : string = "Either a non web service connector has been used or a
            connection has been made which breaks one or more rules"; >> ;
}
```

Appendix E

Complete ACME Descriptions of Enhanced Style Scenarios

E.1 Car Parking Scenario

E.1.1 Initial Configuration

```
import families/ws_enhanced_01.acme;

System ScenarioCarparkInitial : ws_enhanced_01 = new ws_enhanced_01 extended with {

  Component CompTWSAnalysisControl0 : CompTWSAnalysisControl = new CompTWSAnalysisControl extended with {
    Property ActiveAnalysisCentralDataStoreCorrect = true;
    Property ActiveAnalysisCommissionMismatch = true;
    Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
    Property ActiveAnalysisCommissionPartialMatch = true;
    Property ActiveAnalysisConcurrentCallsToThisPort = true;
    Property ActiveAnalysisMessageDataTypesMatch = true;
    Property ActiveAnalysisMessageExchangePatternsMatch = true;
    Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
    Property ActiveAnalysisMessageOverData = true;
    Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }


  Component CPClient : CompTWSClient = new CompTWSClient extended with {
    Port setupConf : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

      Property MessagePattern = "SOLI
        CPClient_setupConf = CPClient_setupConf_sendReq -> CPClient_setupConf_p1
        CPClient_setupConf_p1 = CPClient_setupConf_p2 [] CPClient_setupConf_p3
```

```
        CPClient_setupConf_p2 = CPClient_setupConf_getRes -> CPClient_setupConf_OK
        CPClient_setupConf_p3 = CPClient_setupConf_getFault -> CPClient_setupConf_FAULT
        CPClient_setupConf_OK = CPClient_PaymentCC
        CPClient_setupConf_FAULT = CPClient_PaymentCC";

      Property Messages : TMessages = {
        [MessageId = "CPClient_setupConf_sendReq"; MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_setupConf_getRes";MessageData = {
          [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_setupConf_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property ChoiceGroup = "CarPark";
      Property DataContinuity = Sporadic;
      Property GroupChoiceMaker = Yes;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port PaymentCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

      Property MessagePattern = "SOLI
        CPClient_PaymentCC = CPClient_PaymentCC_sendReq -> CPClient_PaymentCC_p1
        CPClient_PaymentCC_p1 = CPClient_PaymentCC_p2 [] CPClient_PaymentCC_p3
        CPClient_PaymentCC_p2 = CPClient_PaymentCC_getRes -> CPClient_PaymentCC_OK
        CPClient_PaymentCC_p3 = CPClient_PaymentCC_getFault -> CPClient_PaymentCC_FAULT
        CPClient_PaymentCC_OK = CPClient_logout
        CPClient_PaymentCC_FAULT = CPClient_logout";

      Property Messages : TMessages = {
        [MessageId = "CPClient_PaymentCC_sendReq";MessageData = {
          [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
          [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private
              ;]};],
        [MessageId = "CPClient_PaymentCC_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_PaymentCC_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

```

```
    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
    Property BindingSelfRemove = Yes;
    Property ChoiceGroup = "CarPark";
    Property DataContinuity = Sporadic;
    Property GroupChoiceMaker = No;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = Yes;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
  }

  Port logout : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

    Property MessagePattern = "SOLI
      CPClient_logout = CPClient_logout_sendReq -> CPClient_logout_p1
      CPClient_logout_p1 = CPClient_logout_p2 [] CPClient_logout_p3
      CPClient_logout_p2 = CPClient_logout_getRes -> CPClient_logout_OK
      CPClient_logout_p3 = CPClient_logout_getFault -> CPClient_logout_FAULT
      CPClient_logout_OK = CPClient_Thread
      CPClient_logout_FAULT = CPClient_Thread";

    Property Messages : TMessages = {
      [MessageId = "CPClient_logout_sendReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_logout_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_logout_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property SendsFirstMessage = Yes;
    Property Reentrant = No;
    Property InOurControlDomain = Yes;
    Property GroupChoiceMaker = No;
    Property DataContinuity = Sporadic;
    Property ChoiceGroup = "CarPark";
    Property BindingSelfRemove = Yes;
    Property BindingSelfAdd = Yes;
    Property BindingOtherRemove = No;
    Property BindingOtherAdd = No;
    Property BindTime = Instantiation;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
  }

```

```
  Property CentralDataRecords : Set {TCentralDataRecord} = {
    [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
    [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
    [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
    [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
        ;],
    [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
    [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
        Private;],
    [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
    [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited =
        Private;],
    [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

  Property CentralProcessDescription = "CPClient = CPClient_Thread
                                        CPClient_Thread = CPClient_setupConf";

  Property ComponentInOurControlDomain = Yes;
}


Component BookPayCC : CompTWSService = new CompTWSService extended with {
  Port setupConf : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      BookPayCC_setupConf = BookPayCC_setupConf_sendReq -> BookPayCC_setupConf_p1
      BookPayCC_setupConf_p1 = BookPayCC_setupConf_p2 [] BookPayCC_setupConf_p3
      BookPayCC_setupConf_p2 = BookPayCC_setupConf_getRes -> BookPayCC_setupConf_OK
      BookPayCC_setupConf_p3 = BookPayCC_setupConf_getFault -> BookPayCC_setupConf_FAULT
      BookPayCC_setupConf_OK = BookPayCC_PaymentCC
      BookPayCC_setupConf_FAULT = BookPayCC_PaymentCC";

    Property Messages : TMessages = {
      [MessageId = "BookPayCC_setupConf_sendReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "BookPayCC_setupConf_getRes";MessageData = {
        [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "BookPayCC_setupConf_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Run;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = No;
    Property Reentrant = Yes;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/setupConf"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
```

```
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
  }

  Port PaymentCC : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      BookPayCC_PaymentCC = BookPayCC_PaymentCC_sendReq -> BookPayCC_PaymentCC_p1
      BookPayCC_PaymentCC_p1 = BookPayCC_PaymentCC_p2 [] BookPayCC_PaymentCC_p3
      BookPayCC_PaymentCC_p2 = BookPayCC_PaymentCC_getRes -> BookPayCC_PaymentCC_OK
      BookPayCC_PaymentCC_p3 = BookPayCC_PaymentCC_getFault -> BookPayCC_PaymentCC_FAULT
      BookPayCC_PaymentCC_OK = BookPayCC_logout
      BookPayCC_PaymentCC_FAULT = BookPayCC_logout";

    Property Messages : TMessages = {
      [MessageId = "BookPayCC_PaymentCC_sendReq";MessageData = {
        [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
        [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
      [MessageId = "BookPayCC_PaymentCC_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "BookPayCC_PaymentCC_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property SendsFirstMessage = No;
    Property Reentrant = Yes;
    Property InOurControlDomain = No;
    Property DataContinuity = Sporadic;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Run;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/PaymentCC"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
  }

  Port logout : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
     BookPayCC_logout = BookPayCC_logout_sendReq -> BookPayCC_logout_p1
     BookPayCC_logout_p1 = BookPayCC_logout_p2 [] BookPayCC_logout_p3
     BookPayCC_logout_p2 = BookPayCC_logout_getRes -> BookPayCC_logout_OK
     BookPayCC_logout_p3 = BookPayCC_logout_getFault -> BookPayCC_logout_FAULT
     BookPayCC_logout_OK = BookPayCC_Thread
     BookPayCC_logout_FAULT = BookPayCC_Thread";
```

```
    Property Messages : TMessages = {
      [MessageId = "BookPayCC_logout_sendReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "BookPayCC_logout_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "BookPayCC_logout_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property SendsFirstMessage = No;
    Property Reentrant = No;
    Property InOurControlDomain = No;
    Property DataContinuity = Sporadic;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Run;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/logout"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
  }

  Property CentralProcessDescription = "BookPayCC = BookPayCC_Thread
                                      BookPayCC_Thread = BookPayCC_setupConf";

  Property CentralDataRecords : Set {TCentralDataRecord} = {
    [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
    [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
    [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
    [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private;],
    [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
    [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
        Private;],
    [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
    [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited =
        Private;],
    [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

   Property ComponentInOurControlDomain = No;
 }


 Component SpaceCCBuy : CompTWSService = new CompTWSService extended with {
   Port login : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

     Property MessagePattern = "REQR
       SpaceCCBuy_login = SpaceCCBuy_login_sendReq -> SpaceCCBuy_login_p1
       SpaceCCBuy_login_p1 = SpaceCCBuy_login_p2 [] SpaceCCBuy_login_p3
       SpaceCCBuy_login_p2 = SpaceCCBuy_login_getRes -> SpaceCCBuy_login_OK
```

```
      SpaceCCBuy_login_p3 = SpaceCCBuy_login_getFault -> SpaceCCBuy_login_FAULT
      SpaceCCBuy_login_OK = SpaceCCBuy_checkCreditCard
      SpaceCCBuy_login_FAULT = SpaceCCBuy_checkCreditCard";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_login_sendReq";MessageData = {
        [DatumId = "user";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_login_getRes";MessageData = {
        [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_login_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = No;
    Property Reentrant = Yes;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/login"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
  }

  Port checkCreditCard : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      SpaceCCBuy_checkCreditCard = SpaceCCBuy_checkCreditCard_sendReq ->
          SpaceCCBuy_checkCreditCard_p1
      SpaceCCBuy_checkCreditCard_p1 = SpaceCCBuy_checkCreditCard_p2 []
          SpaceCCBuy_checkCreditCard_p3
      SpaceCCBuy_checkCreditCard_p2 = SpaceCCBuy_checkCreditCard_getRes ->
          SpaceCCBuy_checkCreditCard_OK
      SpaceCCBuy_checkCreditCard_p3 = SpaceCCBuy_checkCreditCard_getFault ->
          SpaceCCBuy_checkCreditCard_FAULT
      SpaceCCBuy_checkCreditCard_OK = SpaceCCBuy_payByCC
      SpaceCCBuy_checkCreditCard_FAULT = SpaceCCBuy_payByCC";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_checkCreditCard_sendReq";MessageData = {
        [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "cardNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "expDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_checkCreditCard_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_checkCreditCard_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};
```

```
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property InOurControlDomain = No;
    Property DataContinuity = Sporadic;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Instantiation;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/checkCreditCard"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
  }

  Port payByCC : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      SpaceCCBuy_payByCC = SpaceCCBuy_payByCC_sendReq -> SpaceCCBuy_payByCC_p1
      SpaceCCBuy_payByCC_p1 = SpaceCCBuy_payByCC_p2 [] SpaceCCBuy_payByCC_p3
      SpaceCCBuy_payByCC_p2 = SpaceCCBuy_payByCC_getRes -> SpaceCCBuy_payByCC_OK
      SpaceCCBuy_payByCC_p3 = SpaceCCBuy_payByCC_getFault -> SpaceCCBuy_payByCC_FAULT
      SpaceCCBuy_payByCC_OK = SpaceCCBuy_logout
      SpaceCCBuy_payByCC_FAULT = SpaceCCBuy_logout";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_payByCC_sendReq";MessageData = {
        [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_payByCC_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_payByCC_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = No;
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/payByCC"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
  }
```

```
  Port logout : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      SpaceCCBuy_logout = SpaceCCBuy_logout_sendReq -> SpaceCCBuy_logout_p1
      SpaceCCBuy_logout_p1 = SpaceCCBuy_logout_p2 [] SpaceCCBuy_logout_p3
      SpaceCCBuy_logout_p2 = SpaceCCBuy_logout_getRes -> SpaceCCBuy_logout_OK
      SpaceCCBuy_logout_p3 = SpaceCCBuy_logout_getFault -> SpaceCCBuy_logout_FAULT
      SpaceCCBuy_logout_OK = SpaceCCBuy_Thread
      SpaceCCBuy_logout_FAULT = SpaceCCBuy_Thread";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_logout_sendReq";MessageData = {
        [DatumId = "user";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_logout_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_logout_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property InOurControlDomain = No;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Instantiation;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/logout"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
    Property DataContinuity = Sporadic;
  }

  Property CentralProcessDescription = "SpaceCCBuy = SpaceCCBuy_Thread
                                       SpaceCCBuy_Thread = SpaceCCBuy_login";

  Property CentralDataRecords : Set {TCentralDataRecord} = {
    [DatumID = "user";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
    [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
    [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
    [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
         ;],
    [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
    [DatumID = "cardNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
         Private;],
    [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
    [DatumID = "expDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited = Private
         ;],
    [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

```

```
    Property ComponentInOurControlDomain = No;
  }

  Connector ConnTWS0 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS1 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS2 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS3 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS4 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS5 : ConnTWS = new ConnTWS extended with {  }

  Connector ConnTWSStubborn0 : ConnTWSStubborn = new ConnTWSStubborn extended with {  }

  Attachment BookPayCC.setupConf to ConnTWS0.role1;
  Attachment CPClient.setupConf to ConnTWS0.role2;
  Attachment BookPayCC.PaymentCC to ConnTWS1.role2;
  Attachment CPClient.PaymentCC to ConnTWS1.role1;
  Attachment CPClient.logout to ConnTWS2.role1;
  Attachment BookPayCC.logout to ConnTWS2.role2;
  Attachment SpaceCCBuy.login to ConnTWS3.role2;
  Attachment CPClient.setupConf to ConnTWS3.role1;
  Attachment SpaceCCBuy.checkCreditCard to ConnTWS4.role2;
  Attachment CPClient.PaymentCC to ConnTWS4.role1;
  Attachment CPClient.logout to ConnTWS5.role1;
  Attachment SpaceCCBuy.logout to ConnTWS5.role2;
  Attachment SpaceCCBuy.payByCC to ConnTWSStubborn0.role1;
}
```

E.1.2 Final Configuration

```
import families/ws_enhanced_01.acme;

System ScenarioCarparkFinal : ws_enhanced_01 = new ws_enhanced_01 extended with {

  Component CompTWSAnalysisControl0 : CompTWSAnalysisControl = new CompTWSAnalysisControl
      extended with {
    Property ActiveAnalysisCentralDataStoreCorrect = true;
    Property ActiveAnalysisCommissionMismatch = true;
    Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
    Property ActiveAnalysisCommissionPartialMatch = true;
    Property ActiveAnalysisConcurrentCallsToThisPort = true;
    Property ActiveAnalysisMessageDataTypesMatch = true;
    Property ActiveAnalysisMessageExchangePatternsMatch = true;
    Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
    Property ActiveAnalysisMessageOverData = true;
    Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }


```

```
Component CPClient : CompTWSClient = new CompTWSClient extended with {
  Port setupConf : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

    Property MessagePattern = "SOLI
      CPClient_setupConf = CPClient_setupConf_sendReq -> CPClient_setupConf_p1
      CPClient_setupConf_p1 = CPClient_setupConf_p2 [] CPClient_setupConf_p3
      CPClient_setupConf_p2 = CPClient_setupConf_getRes -> CPClient_setupConf_OK
      CPClient_setupConf_p3 = CPClient_setupConf_getFault -> CPClient_setupConf_FAULT
      CPClient_setupConf_OK = CPClient_PaymentCC
      CPClient_setupConf_FAULT = CPClient_PaymentCC";

    Property Messages : TMessages = {
      [MessageId = "CPClient_setupConf_sendReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_setupConf_getRes";MessageData = {
        [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_setupConf_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
    Property BindingSelfRemove = Yes;
    Property ChoiceGroup = "CarPark";
    Property DataContinuity = Sporadic;
    Property GroupChoiceMaker = Yes;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = Yes;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
  }

  Port PaymentCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

    Property MessagePattern = "SOLI
      CPClient_PaymentCC = CPClient_PaymentCC_sendReq -> CPClient_PaymentCC_p1
      CPClient_PaymentCC_p1 = CPClient_PaymentCC_p2 [] CPClient_PaymentCC_p3
      CPClient_PaymentCC_p2 = CPClient_PaymentCC_getRes -> CPClient_PaymentCC_OK
      CPClient_PaymentCC_p3 = CPClient_PaymentCC_getFault -> CPClient_PaymentCC_FAULT
      CPClient_PaymentCC_OK = CPClient_logout
      CPClient_PaymentCC_FAULT = CPClient_logout";

    Property Messages : TMessages = {
      [MessageId = "CPClient_PaymentCC_sendReq";MessageData = {
        [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
```

```
        [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_PaymentCC_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_PaymentCC_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
    Property BindingSelfRemove = Yes;
    Property ChoiceGroup = "CarPark";
    Property DataContinuity = Sporadic;
    Property GroupChoiceMaker = No;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = Yes;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
  }

  Port logout : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

    Property MessagePattern = "SOLI
      CPClient_logout = CPClient_logout_sendReq -> CPClient_logout_p1
      CPClient_logout_p1 = CPClient_logout_p2 [] CPClient_logout_p3
      CPClient_logout_p2 = CPClient_logout_getRes -> CPClient_logout_OK
      CPClient_logout_p3 = CPClient_logout_getFault -> CPClient_logout_FAULT
      CPClient_logout_OK = CPClient_Thread
      CPClient_logout_FAULT = CPClient_Thread";

    Property Messages : TMessages = {
      [MessageId = "CPClient_logout_sendReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_logout_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_logout_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property SendsFirstMessage = Yes;
    Property Reentrant = No;
    Property InOurControlDomain = Yes;
    Property GroupChoiceMaker = No;
    Property DataContinuity = Sporadic;
    Property ChoiceGroup = "CarPark";
    Property BindingSelfRemove = Yes;
    Property BindingSelfAdd = Yes;
    Property BindingOtherRemove = No;
    Property BindingOtherAdd = No;
    Property BindTime = Instantiation;
```

```
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
      [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
      [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private;],
      [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
      [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited = Private;],
      [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
      [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited = Private;],
      [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

    Property CentralProcessDescription = "CPClient = CPClient_Thread
                                          CPClient_Thread = CPClient_setupConf";

    Property ComponentInOurControlDomain = Yes;
  }

  Component BookPayCC : CompTWSService = new CompTWSService extended with {
    Port setupConf : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

      Property MessagePattern = "REQR
        BookPayCC_setupConf = BookPayCC_setupConf_sendReq -> BookPayCC_setupConf_p1
        BookPayCC_setupConf_p1 = BookPayCC_setupConf_p2 [] BookPayCC_setupConf_p3
        BookPayCC_setupConf_p2 = BookPayCC_setupConf_getRes -> BookPayCC_setupConf_OK
        BookPayCC_setupConf_p3 = BookPayCC_setupConf_getFault -> BookPayCC_setupConf_FAULT
        BookPayCC_setupConf_OK = BookPayCC_PaymentCC
        BookPayCC_setupConf_FAULT = BookPayCC_PaymentCC";

      Property Messages : TMessages = {
        [MessageId = "BookPayCC_setupConf_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_setupConf_getRes";MessageData = {
          [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_setupConf_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Run;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = No;
```

```
      Property Reentrant = Yes;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/setupConf"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
    }

    Port PaymentCC : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

      Property MessagePattern = "REQR
        BookPayCC_PaymentCC = BookPayCC_PaymentCC_sendReq -> BookPayCC_PaymentCC_p1
        BookPayCC_PaymentCC_p1 = BookPayCC_PaymentCC_p2 [] BookPayCC_PaymentCC_p3
        BookPayCC_PaymentCC_p2 = BookPayCC_PaymentCC_getRes -> BookPayCC_PaymentCC_OK
        BookPayCC_PaymentCC_p3 = BookPayCC_PaymentCC_getFault -> BookPayCC_PaymentCC_FAULT
        BookPayCC_PaymentCC_OK = BookPayCC_logout
        BookPayCC_PaymentCC_FAULT = BookPayCC_logout";

      Property Messages : TMessages = {
        [MessageId = "BookPayCC_PaymentCC_sendReq";MessageData = {
          [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
          [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private
              ;]};],
        [MessageId = "BookPayCC_PaymentCC_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_PaymentCC_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property SendsFirstMessage = No;
      Property Reentrant = Yes;
      Property InOurControlDomain = No;
      Property DataContinuity = Sporadic;
      Property BindingSelfRemove = No;
      Property BindingSelfAdd = No;
      Property BindingOtherRemove = Yes;
      Property BindingOtherAdd = Yes;
      Property BindTime = Run;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/PaymentCC"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
    }

    Port logout : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

      Property MessagePattern = "REQR
```

```
        BookPayCC_logout = BookPayCC_logout_sendReq -> BookPayCC_logout_p1
        BookPayCC_logout_p1 = BookPayCC_logout_p2 [] BookPayCC_logout_p3
        BookPayCC_logout_p2 = BookPayCC_logout_getRes -> BookPayCC_logout_OK
        BookPayCC_logout_p3 = BookPayCC_logout_getFault -> BookPayCC_logout_FAULT
        BookPayCC_logout_OK = BookPayCC_Thread
        BookPayCC_logout_FAULT = BookPayCC_Thread";

      Property Messages : TMessages = {
        [MessageId = "BookPayCC_logout_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_logout_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_logout_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property SendsFirstMessage = No;
      Property Reentrant = No;
      Property InOurControlDomain = No;
      Property DataContinuity = Sporadic;
      Property BindingSelfRemove = No;
      Property BindingSelfAdd = No;
      Property BindingOtherRemove = Yes;
      Property BindingOtherAdd = Yes;
      Property BindTime = Run;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/logout"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
    }

    Property CentralProcessDescription = "BookPayCC = BookPayCC_Thread
                                          BookPayCC_Thread = BookPayCC_setupConf";

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
      [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
      [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
          ;],
      [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
      [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
          Private;],
      [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
      [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited =
          Private;],
      [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

      Property ComponentInOurControlDomain = No;
  }
```

```
Component SpaceCCBuy : CompTWSService = new CompTWSService extended with {
  Port login : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      SpaceCCBuy_login = SpaceCCBuy_login_sendReq -> SpaceCCBuy_login_p1
      SpaceCCBuy_login_p1 = SpaceCCBuy_login_p2 [] SpaceCCBuy_login_p3
      SpaceCCBuy_login_p2 = SpaceCCBuy_login_getRes -> SpaceCCBuy_login_OK
      SpaceCCBuy_login_p3 = SpaceCCBuy_login_getFault -> SpaceCCBuy_login_FAULT
      SpaceCCBuy_login_OK = SpaceCCBuy_checkCreditCard
      SpaceCCBuy_login_FAULT = SpaceCCBuy_checkCreditCard";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_login_sendReq";MessageData = {
        [DatumId = "user";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_login_getRes";MessageData = {
        [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_login_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = No;
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/login"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
  }

  Port checkCreditCard : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      SpaceCCBuy_checkCreditCard = SpaceCCBuy_checkCreditCard_sendReq ->
           SpaceCCBuy_checkCreditCard_p1
      SpaceCCBuy_checkCreditCard_p1 = SpaceCCBuy_checkCreditCard_p2 []
           SpaceCCBuy_checkCreditCard_p3
      SpaceCCBuy_checkCreditCard_p2 = SpaceCCBuy_checkCreditCard_getRes ->
           SpaceCCBuy_checkCreditCard_OK
      SpaceCCBuy_checkCreditCard_p3 = SpaceCCBuy_checkCreditCard_getFault ->
           SpaceCCBuy_checkCreditCard_FAULT
      SpaceCCBuy_checkCreditCard_OK = SpaceCCBuy_payByCC
      SpaceCCBuy_checkCreditCard_FAULT = SpaceCCBuy_payByCC";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_checkCreditCard_sendReq";MessageData = {
```

```
        [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "cardNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "expDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_checkCreditCard_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_checkCreditCard_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property InOurControlDomain = No;
    Property DataContinuity = Sporadic;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Instantiation;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/checkCreditCard"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
  }

  Port payByCC : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      SpaceCCBuy_payByCC = SpaceCCBuy_payByCC_sendReq -> SpaceCCBuy_payByCC_p1
      SpaceCCBuy_payByCC_p1 = SpaceCCBuy_payByCC_p2 [] SpaceCCBuy_payByCC_p3
      SpaceCCBuy_payByCC_p2 = SpaceCCBuy_payByCC_getRes -> SpaceCCBuy_payByCC_OK
      SpaceCCBuy_payByCC_p3 = SpaceCCBuy_payByCC_getFault -> SpaceCCBuy_payByCC_FAULT
      SpaceCCBuy_payByCC_OK = SpaceCCBuy_logout
      SpaceCCBuy_payByCC_FAULT = SpaceCCBuy_logout";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_payByCC_sendReq";MessageData = {
        [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_payByCC_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_payByCC_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = No;
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
```

```
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/payByCC"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
  }

  Port logout : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

    Property MessagePattern = "REQR
      SpaceCCBuy_logout = SpaceCCBuy_logout_sendReq -> SpaceCCBuy_logout_p1
      SpaceCCBuy_logout_p1 = SpaceCCBuy_logout_p2 [] SpaceCCBuy_logout_p3
      SpaceCCBuy_logout_p2 = SpaceCCBuy_logout_getRes -> SpaceCCBuy_logout_OK
      SpaceCCBuy_logout_p3 = SpaceCCBuy_logout_getFault -> SpaceCCBuy_logout_FAULT
      SpaceCCBuy_logout_OK = SpaceCCBuy_Thread
      SpaceCCBuy_logout_FAULT = SpaceCCBuy_Thread";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_logout_sendReq";MessageData = {
        [DatumId = "user";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_logout_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_logout_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property InOurControlDomain = No;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Instantiation;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/logout"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
    Property DataContinuity = Sporadic;
  }

  Property CentralProcessDescription = "SpaceCCBuy = SpaceCCBuy_Thread
                                        SpaceCCBuy_Thread = SpaceCCBuy_login";

  Property CentralDataRecords : Set {TCentralDataRecord} = {
    [DatumID = "user";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
    [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
    [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
    [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private;],
    [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
```

```
      [DatumID = "cardNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
          Private;],
      [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
      [DatumID = "expDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited = Private
          ;],
      [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

    Property ComponentInOurControlDomain = No;
  }


  Component SCENE_Framework : CompTWSIntermediary = new CompTWSIntermediary extended with {
    Port In_login : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

      Property MessagePattern = "REQR
        SCENE_Framework_In_login = SCENE_Framework_In_login_getReq -> SCENE_Framework_Out_login
        SCENE_Framework_In_login_p1 = SCENE_Framework_In_login_p2 [] SCENE_Framework_In_login_p3
        SCENE_Framework_In_login_p2 = SCENE_Framework_In_login_sendRes ->
            SCENE_Framework_In_login_OK
        SCENE_Framework_In_login_p3 = SCENE_Framework_In_login_sendFault ->
            SCENE_Framework_In_login_FAULT
        SCENE_Framework_In_login_OK = SCENE_Framework_In_PaymentCC
        SCENE_Framework_In_login_FAULT = SCENE_Framework_In_PaymentCC";

      Property Messages : TMessages = {
        [MessageId = "SCENE_Framework_In_login_getReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "SCENE_Framework_In_login_sendRes";MessageData = {
          [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "SCENE_Framework_In_login_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Run;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"192.168.0.1/In_Login"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"192.168.0.1/WSDL"};
    }

    Port In_PaymentCC : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

      Property MessagePattern = "REQR
```

```
    SCENE_Framework_In_PaymentCC = SCENE_Framework_In_PaymentCC_getReq ->
        SCENE_Framework_Process_Branch
    SCENE_Framework_In_PaymentCC_p1 = SCENE_Framework_In_PaymentCC_p2 []
        SCENE_Framework_In_PaymentCC_p3
    SCENE_Framework_In_PaymentCC_p2 = SCENE_Framework_In_PaymentCC_sendRes ->
        SCENE_Framework_In_PaymentCC_OK
    SCENE_Framework_In_PaymentCC_p3 = SCENE_Framework_In_PaymentCC_sendFault ->
        SCENE_Framework_In_PaymentCC_FAULT
    SCENE_Framework_In_PaymentCC_OK = SCENE_Framework_In_logout
    SCENE_Framework_In_PaymentCC_FAULT = SCENE_Framework_In_logout";

  Property Messages : TMessages = {
    [MessageId = "SCENE_Framework_In_PaymentCC_getReq";MessageData = {
      [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
      [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
      [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
      [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private
          ;]};],
    [MessageId = "SCENE_Framework_In_PaymentCC_sendRes";MessageData = {
      [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
    [MessageId = "SCENE_Framework_In_PaymentCC_sendFault";MessageData = {
      [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

  Property BindTime = Run;
  Property BindingOtherAdd = Yes;
  Property BindingOtherRemove = Yes;
  Property BindingSelfAdd = No;
  Property BindingSelfRemove = No;
  Property DataContinuity = Sporadic;
  Property InOurControlDomain = Yes;
  Property Reentrant = No;
  Property SendsFirstMessage = No;
  Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
  Property EndPointAddressList : TEndPointAddresses = {"192.168.0.1/In_PaymentCC"};
  Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
      LateTimingFailures,HaltFailures,ErraticFailures};
  Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
      LateTimingFailures,HaltFailures,ErraticFailures};
  Property WsdlDocRefs : TWsdlDocs = {"192.168.0.1/WSDL"};
}

Port In_logout : PortTWSServiceSingle = new PortTWSServiceSingle extended with {

  Property MessagePattern = "REQR
    SCENE_Framework_In_logout = SCENE_Framework_In_logout_getReq ->
        SCENE_Framework_Out_logout
    SCENE_Framework_In_logout_p1 = SCENE_Framework_In_logout_p2 []
        SCENE_Framework_In_logout_p3
    SCENE_Framework_In_logout_p2 = SCENE_Framework_In_logout_sendRes ->
        SCENE_Framework_In_logout_OK
    SCENE_Framework_In_logout_p3 = SCENE_Framework_In_logout_sendFault ->
        SCENE_Framework_In_logout_FAULT
    SCENE_Framework_In_logout_OK = SCENE_Framework_Thread
    SCENE_Framework_In_logout_FAULT = SCENE_Framework_Thread";
```

```

    Property Messages : TMessages = {
      [MessageId = "SCENE_Framework_In_logout_getReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SCENE_Framework_In_logout_sendRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SCENE_Framework_In_logout_sendFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property SendsFirstMessage = No;
    Property Reentrant = No;
    Property InOurControlDomain = Yes;
    Property DataContinuity = Sporadic;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Run;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"192.168.0.1/In_Logout"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"192.168.0.1/WSDL"};
  }

  Port Out_login : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

    Property MessagePattern = "SOLI
      SCENE_Framework_Out_login = SCENE_Framework_Out_login_sendReq ->
           SCENE_Framework_Out_login_p1
      SCENE_Framework_Out_login_p1 = SCENE_Framework_Out_login_p2 []
           SCENE_Framework_Out_login_p3
      SCENE_Framework_Out_login_p2 = SCENE_Framework_Out_login_getRes ->
           SCENE_Framework_Out_login_OK
      SCENE_Framework_Out_login_p3 = SCENE_Framework_Out_login_getFault ->
           SCENE_Framework_Out_login_FAULT
      SCENE_Framework_Out_login_OK = SCENE_Framework_In_login_p2
      SCENE_Framework_Out_login_FAULT = SCENE_Framework_In_login_p3";

    Property Messages : TMessages = {
      [MessageId = "SCENE_Framework_Out_login_sendReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SCENE_Framework_Out_login_getRes";MessageData = {
        [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SCENE_Framework_Out_login_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
```

```
      Property BindingSelfRemove = Yes;
      Property ChoiceGroup = "Service";
      Property DataContinuity = Sporadic;
      Property GroupChoiceMaker = Yes;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port Out_paymentCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

      Property MessagePattern = "SOLI
        SCENE_Framework_Out_PaymentCC = SCENE_Framework_Out_PaymentCC_sendReq ->
            SCENE_Framework_Out_PaymentCC_p1
        SCENE_Framework_Out_PaymentCC_p1 = SCENE_Framework_Out_PaymentCC_p2 []
            SCENE_Framework_Out_PaymentCC_p3
        SCENE_Framework_Out_PaymentCC_p2 = SCENE_Framework_Out_PaymentCC_getRes ->
            SCENE_Framework_Out_PaymentCC_OK
        SCENE_Framework_Out_PaymentCC_p3 = SCENE_Framework_Out_PaymentCC_getFault ->
            SCENE_Framework_Out_PaymentCC_FAULT
        SCENE_Framework_Out_PaymentCC_OK = SCENE_Framework_In_PaymentCC_p2
        SCENE_Framework_Out_PaymentCC_FAULT = SCENE_Framework_In_PaymentCC_p3";

      Property Messages : TMessages = {
        [MessageId = "SCENE_Framework_Out_PaymentCC_sendReq";MessageData = {
          [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
          [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private
              ;]};],
        [MessageId = "SCENE_Framework_Out_PaymentCC_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "SCENE_Framework_Out_PaymentCC_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property ChoiceGroup = "Service";
      Property DataContinuity = Sporadic;
      Property GroupChoiceMaker = No;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
```

```
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port Out_checkCreditCard : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

      Property MessagePattern = "SOLI
        SCENE_Framework_Out_checkCreditCard = SCENE_Framework_Out_checkCreditCard_sendReq -> SCENE_Framework_Out_checkCreditCard_p1
        SCENE_Framework_Out_checkCreditCard_p1 = SCENE_Framework_Out_checkCreditCard_p2 [] SCENE_Framework_Out_checkCreditCard_p3
        SCENE_Framework_Out_checkCreditCard_p2 = SCENE_Framework_Out_checkCreditCard_getRes -> SCENE_Framework_Out_checkCreditCard_OK
        SCENE_Framework_Out_checkCreditCard_p3 = SCENE_Framework_Out_checkCreditCard_getFault -> SCENE_Framework_Out_checkCreditCard_FAULT
        SCENE_Framework_Out_checkCreditCard_OK = SCENE_Framework_Out_payByCC
        SCENE_Framework_Out_checkCreditCard_FAULT = SCENE_Framework_Out_payByCC";

      Property Messages : TMessages = {
        [MessageId = "SCENE_Framework_Out_checkCreditCard_sendReq";MessageData = {
          [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
        [MessageId = "SCENE_Framework_Out_checkCreditCard_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "SCENE_Framework_Out_checkCreditCard_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property InOurControlDomain = Yes;
      Property DataContinuity = Sporadic;
      Property BindingSelfRemove = Yes;
      Property BindingSelfAdd = Yes;
      Property GroupChoiceMaker = No;
      Property ChoiceGroup = "Service";
      Property BindingOtherRemove = No;
      Property BindingOtherAdd = No;
      Property BindTime = Instantiation;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port Out_payByCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

      Property MessagePattern = "SOLI
        SCENE_Framework_Out_payByCC = SCENE_Framework_Out_payByCC_sendReq -> SCENE_Framework_Out_payByCC_p1
        SCENE_Framework_Out_payByCC_p1 = SCENE_Framework_Out_payByCC_p2 [] SCENE_Framework_Out_payByCC_p3
```

```
      SCENE_Framework_Out_payByCC_p2 = SCENE_Framework_Out_payByCC_getRes ->
          SCENE_Framework_Out_payByCC_OK
      SCENE_Framework_Out_payByCC_p3 = SCENE_Framework_Out_payByCC_getFault ->
          SCENE_Framework_Out_payByCC_FAULT
      SCENE_Framework_Out_payByCC_OK = SCENE_Framework_In_PaymentCC_p2
      SCENE_Framework_Out_payByCC_FAULT = SCENE_Framework_In_PaymentCC_p3";

    Property Messages : TMessages = {
      [MessageId = "SCENE_Framework_Out_payByCC_sendReq";MessageData = {
        [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;]};],
      [MessageId = "SCENE_Framework_Out_payByCC_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SCENE_Framework_Out_payByCC_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
    Property BindingSelfRemove = Yes;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = Yes;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property ChoiceGroup = "Service";
    Property GroupChoiceMaker = No;
  }

  Port Out_logout : PortTWSClientUnicast = new PortTWSClientUnicast extended with {

    Property MessagePattern = "SOLI
      SCENE_Framework_Out_logout = SCENE_Framework_Out_logout_sendReq ->
          SCENE_Framework_Out_logout_p1
      SCENE_Framework_Out_logout_p1 = SCENE_Framework_Out_logout_p2 []
          SCENE_Framework_Out_logout_p3
      SCENE_Framework_Out_logout_p2 = SCENE_Framework_Out_logout_getRes ->
          SCENE_Framework_Out_logout_OK
      SCENE_Framework_Out_logout_p3 = SCENE_Framework_Out_logout_getFault ->
          SCENE_Framework_Out_logout_FAULT
      SCENE_Framework_Out_logout_OK = SCENE_Framework_In_logout_p2
      SCENE_Framework_Out_logout_FAULT = SCENE_Framework_In_logout_p3";

    Property Messages : TMessages = {
      [MessageId = "SCENE_Framework_Out_logout_sendReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SCENE_Framework_Out_logout_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SCENE_Framework_Out_logout_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};
```

```

      Property SendsFirstMessage = Yes;
      Property Reentrant = No;
      Property InOurControlDomain = Yes;
      Property GroupChoiceMaker = No;
      Property DataContinuity = Sporadic;
      Property ChoiceGroup = "Service";
      Property BindingSelfRemove = Yes;
      Property BindingSelfAdd = Yes;
      Property BindingOtherRemove = No;
      Property BindingOtherAdd = No;
      Property BindTime = Instantiation;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
      [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
      [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
          ;],
      [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
      [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
          Private;],
      [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
      [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited =
          Private;],
      [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

    Property CentralProcessDescription = "SCENE_Framework = SCENE_Framework_Thread
                                          SCENE_Framework_Thread = SCENE_Framework_In_login
                                          SCENE_Framework_Process_Branch =
                                              SCENE_Framework_Out_PaymentCC []
                                              SCENE_Framework_Out_checkCreditCard";

    Property ComponentInOurControlDomain = Yes;
  }

  Connector ConnTWS0 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS1 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS2 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS3 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS4 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS5 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS6 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS7 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS8 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS9 : ConnTWS = new ConnTWS extended with {  }

  Attachment CPClient.setupConf to ConnTWS0.role2;
```

```
  Attachment SCENE_Framework.In_login to ConnTWS0.role1;
  Attachment CPClient.PaymentCC to ConnTWS1.role1;
  Attachment SCENE_Framework.In_PaymentCC to ConnTWS1.role2;
  Attachment CPClient.logout to ConnTWS2.role1;
  Attachment SCENE_Framework.In_logout to ConnTWS2.role2;
  Attachment SCENE_Framework.Out_login to ConnTWS3.role1;
  Attachment BookPayCC.setupConf to ConnTWS3.role2;
  Attachment BookPayCC.PaymentCC to ConnTWS4.role2;
  Attachment SCENE_Framework.Out_paymentCC to ConnTWS4.role1;
  Attachment SCENE_Framework.Out_logout to ConnTWS5.role1;
  Attachment BookPayCC.logout to ConnTWS5.role2;
  Attachment SCENE_Framework.Out_login to ConnTWS6.role1;
  Attachment SpaceCCBuy.login to ConnTWS6.role2;
  Attachment SCENE_Framework.Out_checkCreditCard to ConnTWS7.role2;
  Attachment SpaceCCBuy.checkCreditCard to ConnTWS7.role1;
  Attachment SCENE_Framework.Out_payByCC to ConnTWS8.role2;
  Attachment SpaceCCBuy.payByCC to ConnTWS8.role1;
  Attachment SCENE_Framework.Out_logout to ConnTWS9.role2;
  Attachment SpaceCCBuy.logout to ConnTWS9.role1;
}
```

E.2 Additional Tests

E.2.1 Omission Check

```
import families/ws_enhanced_01.acme;

System AdditionalTestOmission : ws_enhanced_01 = new ws_enhanced_01 extended with {

  Component Client : CompTWSClient = new CompTWSClient extended with {
    Port p1 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Client_p1 = Client_p1_sendReq -> Client_p1_p1
        Client_p1_p1 = Client_p1_p2 [] Client_p1_p3
        Client_p1_p2 = Client_p1_getRes -> Client_p1_OK
        Client_p1_p3 = Client_p1_getFault -> Client_p1_FAULT
        Client_p1_OK = Client_p4
        Client_p1_FAULT = Client_p4";

      Property Messages : TMessages = {
        [MessageId = "Client_p1_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p1_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p1_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
```

```
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port p2 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Client_p2 = Client_p2_sendReq -> Client_p2_p1
        Client_p2_p1 = Client_p2_p2 [] Client_p2_p3
        Client_p2_p2 = Client_p2_getRes -> Client_p2_OK
        Client_p2_p3 = Client_p2_getFault -> Client_p2_FAULT
        Client_p2_OK = Client_Thread
        Client_p2_FAULT = Client_Thread";

      Property Messages : TMessages = {
        [MessageId = "Client_p2_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p2_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p2_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port p3 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Client_p3 = Client_p3_sendReq -> Client_p3_p1
        Client_p3_p1 = Client_p3_p2 [] Client_p3_p3
        Client_p3_p2 = Client_p3_getRes -> Client_p3_OK
        Client_p3_p3 = Client_p3_getFault -> Client_p3_FAULT
        Client_p3_OK = Client_Thread
        Client_p3_FAULT = Client_Thread";

      Property Messages : TMessages = {
        [MessageId = "Client_p3_sendReq";MessageData = {
```

```
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p3_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p3_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port p4 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Client_p4 = Client_p4_sendReq -> Client_p4_p1
        Client_p4_p1 = Client_p4_p2 [] Client_p4_p3
        Client_p4_p2 = Client_p4_getRes -> Client_p4_OK
        Client_p4_p3 = Client_p4_getFault -> Client_p4_FAULT
        Client_p4_OK = Client_p2
        Client_p4_FAULT = Client_p2";

      Property Messages : TMessages = {
        [MessageId = "Client_p4_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p4_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p4_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }
```

```

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "sendData";DatumSemantics = "sendData";DatumScopeExhibited = Private;],
      [DatumID = "resultData";DatumSemantics = "resultData";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private;]};

    Property CentralProcessDescription = "Client = Client_Thread
                                          Client_Thread = Client_p1 [] Client_p3";

    Property ComponentInOurControlDomain = Yes;
  }

  Component Service : CompTWSService = new CompTWSService extended with {
    Port p1 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Service_p1 = Service_p1_getReq -> Service_p1_p1
        Service_p1_p1 = Service_p1_p2 [] Service_p1_p3
        Service_p1_p2 = Service_p1_sendRes -> Service_p1_OK
        Service_p1_p3 = Service_p1_sendFault -> Service_p1_FAULT
        Service_p1_OK = Service_p2
        Service_p1_FAULT = Service_p2";

      Property Messages : TMessages = {
        [MessageId = "Service_p1_getReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Service_p1_sendRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Service_p1_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
      Property EndPointAddressList : TEndPointAddresses = {"www.Service.com/p1"};
      Property WsdlDocRefs : TWsdlDocs = {"www.Service.com/WSDL"};
    }

    Port p2 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Service_p2 = Service_p2_getReq -> Service_p2_p1
        Service_p2_p1 = Service_p2_p2 [] Service_p2_p3
        Service_p2_p2 = Service_p2_sendRes -> Service_p2_OK
```

```
      Service_p2_p3 = Service_p2_sendFault -> Service_p2_FAULT
      Service_p2_OK = Service_p3
      Service_p2_FAULT = Service_p3";

    Property Messages : TMessages = {
     [MessageId = "Service_p2_getReq";MessageData = {
       [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p2_sendRes";MessageData = {
        [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p2_sendFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property EndPointAddressList : TEndPointAddresses = {"www.Service.com/p2"};
    Property WsdlDocRefs : TWsdlDocs = {"www.Service.com/WSDL"};
  }

  Port p3 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
    Property MessagePattern = "REQR
      Service_p3 = Service_p3_getReq -> Service_p3_p1
      Service_p3_p1 = Service_p3_p2 [] Service_p3_p3
      Service_p3_p2 = Service_p3_sendRes -> Service_p3_OK
      Service_p3_p3 = Service_p3_sendFault -> Service_p3_FAULT
      Service_p3_OK = Service_Thread
      Service_p3_FAULT = Service_Thread";

    Property Messages : TMessages = {
      [MessageId = "Service_p3_getReq";MessageData = {
        [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p3_sendRes";MessageData = {
        [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p3_sendFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = Yes;
```

```
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property EndPointAddressList : TEndPointAddresses = {"www.Service.com/p3"};
    Property WsdlDocRefs : TWsdlDocs = {"www.Service.com/WSDL"};
  }

  Port p4 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
    Property MessagePattern = "REQR
      Service_p4 = Service_p4_getReq -> Service_p4_p1
      Service_p4_p1 = Service_p4_p2 [] Service_p4_p3
      Service_p4_p2 = Service_p4_sendRes -> Service_p4_OK
      Service_p4_p3 = Service_p4_sendFault -> Service_p4_FAULT
      Service_p4_OK = Service_Thread
      Service_p4_FAULT = Service_Thread";

    Property Messages : TMessages = {
      [MessageId = "Service_p4_getReq";MessageData = {
        [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p4_sendRes";MessageData = {
        [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p4_sendFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    Property EndPointAddressList : TEndPointAddresses = {"www.Service.com/p4"};
    Property WsdlDocRefs : TWsdlDocs = {"www.Service.com/WSDL"};
  }

  Property CentralDataRecords : Set {TCentralDataRecord} = {
    [DatumID = "sendData";DatumSemantics = "sendData";DatumScopeExhibited = Private;],
    [DatumID = "resultData";DatumSemantics = "resultData";DatumScopeExhibited = Private;],
    [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private;]};

  Property CentralProcessDescription = "Service = Service_Thread
                                      Service_Thread = Service_p1 [] Service_p4";
```

```

    Property ComponentInOurControlDomain = Yes;
    }

  Component AnalysisControl : CompTWSAnalysisControl = new CompTWSAnalysisControl extended with {
    Property ActiveAnalysisCentralDataStoreCorrect = true;
    Property ActiveAnalysisCommissionMismatch = true;
    Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
    Property ActiveAnalysisCommissionPartialMatch = true;
    Property ActiveAnalysisConcurrentCallsToThisPort = true;
    Property ActiveAnalysisMessageDataTypesMatch = true;
    Property ActiveAnalysisMessageExchangePatternsMatch = true;
    Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
    Property ActiveAnalysisMessageOverData = true;
    Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }

  Connector ConnTWS0 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWS1 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWS2 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWS3 : ConnTWS = new ConnTWS extended with { }

  Attachment Client.p1 to ConnTWS0.role1;
  Attachment Service.p1 to ConnTWS0.role2;
  Attachment Service.p2 to ConnTWS1.role2;
  Attachment Client.p3 to ConnTWS1.role1;
  Attachment Service.p3 to ConnTWS2.role1;
  Attachment Client.p2 to ConnTWS2.role2;
  Attachment Service.p4 to ConnTWS3.role1;
  Attachment Client.p4 to ConnTWS3.role2;
}
```

E.2.2 Cooperative Connector Check

```
import families/ws_enhanced_01.acme;

System AdditionalTestCooperative : ws_enhanced_01 = new ws_enhanced_01 extended with {

  Component Broker : CompTWSIntermediary = new CompTWSIntermediary extended with {
    Port c1 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Broker_c1 = Broker_c1_sendReq -> Broker_c1_p1
        Broker_c1_p1 = Broker_c1_p2 [] Broker_c1_p3
        Broker_c1_p2 = Broker_c1_getRes -> Broker_c1_OK
        Broker_c1_p3 = Broker_c1_getFault -> Broker_c1_FAULT
        Broker_c1_OK = Broker_s2
        Broker_c1_FAULT = Broker_s2";

```

```
      Property Messages : TMessages = {
        [MessageId = "Broker_c1_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c1_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c1_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
           LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
           LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port c2 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Broker_c2 = Broker_c2_sendReq -> Broker_c2_p1
        Broker_c2_p1 = Broker_c2_p2 [] Broker_c2_p3
        Broker_c2_p2 = Broker_c2_getRes -> Broker_c2_OK
        Broker_c2_p3 = Broker_c2_getFault -> Broker_c2_FAULT
        Broker_c2_OK = Broker_Thread
        Broker_c2_FAULT = Broker_Thread";

      Property Messages : TMessages = {
        [MessageId = "Broker_c2_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c2_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c2_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
           LateTimingFailures,HaltFailures,ErraticFailures};
```

```
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port c3 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Broker_c3 = Broker_c3_sendReq -> Broker_c3_p1
        Broker_c3_p1 = Broker_c3_p2 [] Broker_c3_p3
        Broker_c3_p2 = Broker_c3_getRes -> Broker_c3_OK
        Broker_c3_p3 = Broker_c3_getFault -> Broker_c3_FAULT
        Broker_c3_OK = Broker_Thread
        Broker_c3_FAULT = Broker_Thread";

      Property Messages : TMessages = {
        [MessageId = "Broker_c3_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c3_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c3_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port s1 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Broker_s1 = Broker_s1_getReq -> Broker_s1_p1
        Broker_s1_p1 = Broker_s1_p2 [] Broker_s1_p3
        Broker_s1_p2 = Broker_s1_sendRes -> Broker_s1_OK
        Broker_s1_p3 = Broker_s1_sendFault -> Broker_s1_FAULT
        Broker_s1_OK = Broker_c1
        Broker_s1_FAULT = Broker_c1";

      Property Messages : TMessages = {
        [MessageId = "Broker_s1_getReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s1_sendRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s1_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

```

```
      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property EndPointAddressList : TEndPointAddresses = {"www.Broker.com/s1"};
      Property WsdlDocRefs : TWsdlDocs = {"www.Broker.com/WSDL"};
    }

    Port s2 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Broker_s2 = Broker_s2_getReq -> Broker_s2_p1
        Broker_s2_p1 = Broker_s2_p2 [] Broker_s2_p3
        Broker_s2_p2 = Broker_s2_sendRes -> Broker_s2_OK
        Broker_s2_p3 = Broker_s2_sendFault -> Broker_s2_FAULT
        Broker_s2_OK = Broker_c2
        Broker_s2_FAULT = Broker_c2";

      Property Messages : TMessages = {
        [MessageId = "Broker_s2_getReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s2_sendRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s2_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property EndPointAddressList : TEndPointAddresses = {"www.Broker.com/s2"};
      Property WsdlDocRefs : TWsdlDocs = {"www.Broker.com/WSDL"};
    }

    Port s3 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
```

```
      Property MessagePattern = "REQR
        Broker_s3 = Broker_s3_getReq -> Broker_s3_p1
        Broker_s3_p1 = Broker_s3_p2 [] Broker_s3_p3
        Broker_s3_p2 = Broker_s3_sendRes -> Broker_s3_OK
        Broker_s3_p3 = Broker_s3_sendFault -> Broker_s3_FAULT
        Broker_s3_OK = Broker_Thread
        Broker_s3_FAULT = Broker_Thread";

      Property Messages : TMessages = {
        [MessageId = "Broker_s3_getReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s3_sendRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s3_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property EndPointAddressList : TEndPointAddresses = {"www.Broker.com/s3"};
      Property WsdlDocRefs : TWsdlDocs = {"www.Broker.com/WSDL"};
    }

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "sendData";DatumSemantics = "sendData";DatumScopeExhibited = Private;],
      [DatumID = "resultData";DatumSemantics = "resultData";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
          ;]};

    Property CentralProcessDescription = "Broker = Broker_Thread
                                         Broker_Thread = Broker_s1 [] Broker_s3 [] Broker_c3";

    Property ComponentInOurControlDomain = Yes;
  }

  Component CompTWSAnalysisControl0 : CompTWSAnalysisControl = new CompTWSAnalysisControl
      extended with {
    Property ActiveAnalysisCentralDataStoreCorrect = true;
    Property ActiveAnalysisCommissionMismatch = true;
    Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
    Property ActiveAnalysisCommissionPartialMatch = true;
    Property ActiveAnalysisConcurrentCallsToThisPort = true;
    Property ActiveAnalysisMessageDataTypesMatch = true;
```

```
    Property ActiveAnalysisMessageExchangePatternsMatch = true;
    Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
    Property ActiveAnalysisMessageOverData = true;
    Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }

  Connector ConnTWSCooperative0 : ConnTWSCooperative = new ConnTWSCooperative extended with { }
  Connector ConnTWSCooperative1 : ConnTWSCooperative = new ConnTWSCooperative extended with { }
  Connector ConnTWSCooperative2 : ConnTWSCooperative = new ConnTWSCooperative extended with { }
  Connector ConnTWSCooperative3 : ConnTWSCooperative = new ConnTWSCooperative extended with { }
  Connector ConnTWSCooperative4 : ConnTWSCooperative = new ConnTWSCooperative extended with { }
  Connector ConnTWSCooperative5 : ConnTWSCooperative = new ConnTWSCooperative extended with { }
  Attachment Broker.s1 to ConnTWSCooperative0.role1;
  Attachment Broker.s2 to ConnTWSCooperative1.role1;
  Attachment Broker.s3 to ConnTWSCooperative2.role1;
  Attachment Broker.c1 to ConnTWSCooperative3.role1;
  Attachment Broker.c2 to ConnTWSCooperative4.role1;
  Attachment Broker.c3 to ConnTWSCooperative5.role1;
}
```

E.2.3 Stubborn Connector Check

```
import families/ws_enhanced_01.acme;

System AdditionalTestStubborn : ws_enhanced_01 = new ws_enhanced_01 extended with {

  Component Broker : CompTWSIntermediary = new CompTWSIntermediary extended with {
    Port c1 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Broker_c1 = Broker_c1_sendReq -> Broker_c1_p1
        Broker_c1_p1 = Broker_c1_p2 [] Broker_c1_p3
        Broker_c1_p2 = Broker_c1_getRes -> Broker_c1_OK
        Broker_c1_p3 = Broker_c1_getFault -> Broker_c1_FAULT
        Broker_c1_OK = Broker_s2
        Broker_c1_FAULT = Broker_s2";

      Property Messages : TMessages = {
        [MessageId = "Broker_c1_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c1_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c1_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
```

```
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port c2 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Broker_c2 = Broker_c2_sendReq -> Broker_c2_p1
        Broker_c2_p1 = Broker_c2_p2 [] Broker_c2_p3
        Broker_c2_p2 = Broker_c2_getRes -> Broker_c2_OK
        Broker_c2_p3 = Broker_c2_getFault -> Broker_c2_FAULT
        Broker_c2_OK = Broker_Thread
        Broker_c2_FAULT = Broker_Thread";

      Property Messages : TMessages = {
        [MessageId = "Broker_c2_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c2_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c2_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port c3 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Broker_c3 = Broker_c3_sendReq -> Broker_c3_p1
        Broker_c3_p1 = Broker_c3_p2 [] Broker_c3_p3
        Broker_c3_p2 = Broker_c3_getRes -> Broker_c3_OK
        Broker_c3_p3 = Broker_c3_getFault -> Broker_c3_FAULT
        Broker_c3_OK = Broker_Thread
        Broker_c3_FAULT = Broker_Thread";

```

```
      Property Messages : TMessages = {
        [MessageId = "Broker_c3_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c3_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_c3_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port s1 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Broker_s1 = Broker_s1_getReq -> Broker_s1_p1
        Broker_s1_p1 = Broker_s1_p2 [] Broker_s1_p3
        Broker_s1_p2 = Broker_s1_sendRes -> Broker_s1_OK
        Broker_s1_p3 = Broker_s1_sendFault -> Broker_s1_FAULT
        Broker_s1_OK = Broker_c1
        Broker_s1_FAULT = Broker_c1";

      Property Messages : TMessages = {
        [MessageId = "Broker_s1_getReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s1_sendRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s1_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
```

```
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property EndPointAddressList : TEndPointAddresses = {"www.Broker.com/s1"};
      Property WsdlDocRefs : TWsdlDocs = {"www.Broker.com/WSDL"};
    }

    Port s2 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Broker_s2 = Broker_s2_getReq -> Broker_s2_p1
        Broker_s2_p1 = Broker_s2_p2 [] Broker_s2_p3
        Broker_s2_p2 = Broker_s2_sendRes -> Broker_s2_OK
        Broker_s2_p3 = Broker_s2_sendFault -> Broker_s2_FAULT
        Broker_s2_OK = Broker_c2
        Broker_s2_FAULT = Broker_c2";

      Property Messages : TMessages = {
        [MessageId = "Broker_s2_getReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s2_sendRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s2_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property EndPointAddressList : TEndPointAddresses = {"www.Broker.com/s2"};
      Property WsdlDocRefs : TWsdlDocs = {"www.Broker.com/WSDL"};
    }

    Port s3 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Broker_s3 = Broker_s3_getReq -> Broker_s3_p1
        Broker_s3_p1 = Broker_s3_p2 [] Broker_s3_p3
        Broker_s3_p2 = Broker_s3_sendRes -> Broker_s3_OK
        Broker_s3_p3 = Broker_s3_sendFault -> Broker_s3_FAULT
        Broker_s3_OK = Broker_Thread
        Broker_s3_FAULT = Broker_Thread";

      Property Messages : TMessages = {
        [MessageId = "Broker_s3_getReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s3_sendRes";MessageData = {
```

```
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Broker_s3_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property EndPointAddressList : TEndPointAddresses = {"www.Broker.com/s3"};
      Property WsdlDocRefs : TWsdlDocs = {"www.Broker.com/WSDL"};
    }

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "sendData";DatumSemantics = "sendData";DatumScopeExhibited = Private;],
      [DatumID = "resultData";DatumSemantics = "resultData";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
          ;]};

    Property CentralProcessDescription = "Broker = Broker_Thread
                                       Broker_Thread = Broker_s1 [] Broker_s3 [] Broker_c3";

    Property ComponentInOurControlDomain = Yes;
  }

  Component CompTWSAnalysisControl0 : CompTWSAnalysisControl = new CompTWSAnalysisControl
      extended with {
    Property ActiveAnalysisCentralDataStoreCorrect = true;
    Property ActiveAnalysisCommissionMismatch = true;
    Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
    Property ActiveAnalysisCommissionPartialMatch = true;
    Property ActiveAnalysisConcurrentCallsToThisPort = true;
    Property ActiveAnalysisMessageDataTypesMatch = true;
    Property ActiveAnalysisMessageExchangePatternsMatch = true;
    Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
    Property ActiveAnalysisMessageOverData = true;
    Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }
```

```
  Connector ConnTWSStubborn0 : ConnTWSStubborn = new ConnTWSStubborn extended with { }
  Connector ConnTWSStubborn1 : ConnTWSStubborn = new ConnTWSStubborn extended with { }
  Connector ConnTWSStubborn2 : ConnTWSStubborn = new ConnTWSStubborn extended with { }
  Connector ConnTWSStubborn3 : ConnTWSStubborn = new ConnTWSStubborn extended with { }
  Connector ConnTWSStubborn4 : ConnTWSStubborn = new ConnTWSStubborn extended with { }
  Connector ConnTWSStubborn5 : ConnTWSStubborn = new ConnTWSStubborn extended with { }
  Attachment Broker.s1 to ConnTWSStubborn0.role1;
  Attachment Broker.s2 to ConnTWSStubborn1.role1;
  Attachment Broker.s3 to ConnTWSStubborn2.role1;
  Attachment Broker.c1 to ConnTWSStubborn3.role1;
  Attachment Broker.c2 to ConnTWSStubborn4.role1;
  Attachment Broker.c3 to ConnTWSStubborn5.role1;
}
```

E.2.4 Multiple Connectors Check

E.2.4.1 SpaceCCBuy

```
import families/ws_enhanced_01.acme;
System AdditionalTestMultipleConnectionsSpaceCCBuy : ws_enhanced_01 = new ws_enhanced_01 extended
     with {

  Component CPClient : CompTWSClient = new CompTWSClient extended with {
    Port setupConf : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
      Property MessagePattern = "SOLI
        CPClient_setupConf = CPClient_setupConf_sendReq -> CPClient_setupConf_p1
          CPClient_setupConf_p1 = CPClient_setupConf_p2 [] CPClient_setupConf_p3
          CPClient_setupConf_p2 = CPClient_setupConf_getRes -> CPClient_setupConf_OK
          CPClient_setupConf_p3 = CPClient_setupConf_getFault -> CPClient_setupConf_FAULT
          CPClient_setupConf_OK = CPClient_PaymentCC
          CPClient_setupConf_FAULT = CPClient_PaymentCC";

      Property Messages : TMessages = {
        [MessageId = "CPClient_setupConf_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_setupConf_getRes";MessageData = {
          [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_setupConf_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

        Property BindTime = Instantiation;
        Property BindingOtherAdd = No;
        Property BindingOtherRemove = No;
        Property BindingSelfAdd = Yes;
        Property BindingSelfRemove = Yes;
        Property ChoiceGroup = "CarPark";
        Property DataContinuity = Sporadic;
        Property GroupChoiceMaker = Yes;
        Property InOurControlDomain = Yes;
        Property Reentrant = No;
        Property SendsFirstMessage = Yes;
        Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
```

```
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port PaymentCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
      Property MessagePattern = "SOLI
        CPClient_PaymentCC = CPClient_PaymentCC_sendReq -> CPClient_PaymentCC_p1
        CPClient_PaymentCC_p1 = CPClient_PaymentCC_p2 [] CPClient_PaymentCC_p3
        CPClient_PaymentCC_p2 = CPClient_PaymentCC_getRes -> CPClient_PaymentCC_OK
        CPClient_PaymentCC_p3 = CPClient_PaymentCC_getFault -> CPClient_PaymentCC_FAULT
        CPClient_PaymentCC_OK = CPClient_logout
        CPClient_PaymentCC_FAULT = CPClient_logout";

      Property Messages : TMessages = {
        [MessageId = "CPClient_PaymentCC_sendReq";MessageData = {
          [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
          [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private
              ;]};],
        [MessageId = "CPClient_PaymentCC_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_PaymentCC_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property ChoiceGroup = "CarPark";
      Property DataContinuity = Sporadic;
      Property GroupChoiceMaker = No;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port logout : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
      Property MessagePattern = "SOLI
        CPClient_logout = CPClient_logout_sendReq -> CPClient_logout_p1
        CPClient_logout_p1 = CPClient_logout_p2 [] CPClient_logout_p3
        CPClient_logout_p2 = CPClient_logout_getRes -> CPClient_logout_OK
        CPClient_logout_p3 = CPClient_logout_getFault -> CPClient_logout_FAULT
        CPClient_logout_OK = CPClient_Thread
        CPClient_logout_FAULT = CPClient_Thread";

```

```
      Property Messages : TMessages = {
        [MessageId = "CPClient_logout_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_logout_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_logout_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property SendsFirstMessage = Yes;
      Property Reentrant = No;
      Property InOurControlDomain = Yes;
      Property GroupChoiceMaker = No;
      Property DataContinuity = Sporadic;
      Property ChoiceGroup = "CarPark";
      Property BindingSelfRemove = Yes;
      Property BindingSelfAdd = Yes;
      Property BindingOtherRemove = No;
      Property BindingOtherAdd = No;
      Property BindTime = Instantiation;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
      [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
      [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
          ;],
      [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
      [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
          Private;],
      [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
      [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited =
          Private;],
      [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

    Property CentralProcessDescription = "CPClient = CPClient_Thread
                                          CPClient_Thread = CPClient_setupConf";

    Property ComponentInOurControlDomain = Yes;
  }

  Component SpaceCCBuy : CompTWSService = new CompTWSService extended with {
    Port login : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        SpaceCCBuy_login = SpaceCCBuy_login_sendReq -> SpaceCCBuy_login_p1
        SpaceCCBuy_login_p1 = SpaceCCBuy_login_p2 [] SpaceCCBuy_login_p3
        SpaceCCBuy_login_p2 = SpaceCCBuy_login_getRes -> SpaceCCBuy_login_OK
        SpaceCCBuy_login_p3 = SpaceCCBuy_login_getFault -> SpaceCCBuy_login_FAULT
        SpaceCCBuy_login_OK = SpaceCCBuy_checkCreditCard
```

```
      SpaceCCBuy_login_FAULT = SpaceCCBuy_checkCreditCard";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_login_sendReq";MessageData = {
        [DatumId = "user";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_login_getRes";MessageData = {
        [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_login_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = No;
    Property Reentrant = Yes;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/login"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
  }

  Port checkCreditCard : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
    Property MessagePattern = "REQR
      SpaceCCBuy_checkCreditCard = SpaceCCBuy_checkCreditCard_sendReq ->
          SpaceCCBuy_checkCreditCard_p1
      SpaceCCBuy_checkCreditCard_p1 = SpaceCCBuy_checkCreditCard_p2 []
          SpaceCCBuy_checkCreditCard_p3
      SpaceCCBuy_checkCreditCard_p2 = SpaceCCBuy_checkCreditCard_getRes ->
          SpaceCCBuy_checkCreditCard_OK
      SpaceCCBuy_checkCreditCard_p3 = SpaceCCBuy_checkCreditCard_getFault ->
          SpaceCCBuy_checkCreditCard_FAULT
      SpaceCCBuy_checkCreditCard_OK = SpaceCCBuy_payByCC
      SpaceCCBuy_checkCreditCard_FAULT = SpaceCCBuy_payByCC";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_checkCreditCard_sendReq";MessageData = {
        [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "cardNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "expDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_checkCreditCard_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_checkCreditCard_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property Reentrant = No;
    Property SendsFirstMessage = No;
```

```
      Property InOurControlDomain = No;
      Property DataContinuity = Sporadic;
      Property BindingSelfRemove = No;
      Property BindingSelfAdd = No;
      Property BindingOtherRemove = Yes;
      Property BindingOtherAdd = Yes;
      Property BindTime = Instantiation;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/checkCreditCard"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
    }

    Port payByCC : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        SpaceCCBuy_payByCC = SpaceCCBuy_payByCC_sendReq -> SpaceCCBuy_payByCC_p1
        SpaceCCBuy_payByCC_p1 = SpaceCCBuy_payByCC_p2 [] SpaceCCBuy_payByCC_p3
        SpaceCCBuy_payByCC_p2 = SpaceCCBuy_payByCC_getRes -> SpaceCCBuy_payByCC_OK
        SpaceCCBuy_payByCC_p3 = SpaceCCBuy_payByCC_getFault -> SpaceCCBuy_payByCC_FAULT
        SpaceCCBuy_payByCC_OK = SpaceCCBuy_logout
        SpaceCCBuy_payByCC_FAULT = SpaceCCBuy_logout";

      Property Messages : TMessages = {
        [MessageId = "SpaceCCBuy_payByCC_sendReq";MessageData = {
          [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_payByCC_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_payByCC_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = No;
      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/payByCC"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
    }

    Port logout : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        SpaceCCBuy_logout = SpaceCCBuy_logout_sendReq -> SpaceCCBuy_logout_p1
```

```
      SpaceCCBuy_logout_p1 = SpaceCCBuy_logout_p2 [] SpaceCCBuy_logout_p3
      SpaceCCBuy_logout_p2 = SpaceCCBuy_logout_getRes -> SpaceCCBuy_logout_OK
      SpaceCCBuy_logout_p3 = SpaceCCBuy_logout_getFault -> SpaceCCBuy_logout_FAULT
      SpaceCCBuy_logout_OK = SpaceCCBuy_Thread
      SpaceCCBuy_logout_FAULT = SpaceCCBuy_Thread";

    Property Messages : TMessages = {
      [MessageId = "SpaceCCBuy_logout_sendReq";MessageData = {
        [DatumId = "user";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_logout_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "SpaceCCBuy_logout_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property InOurControlDomain = No;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Instantiation;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/logout"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
    Property DataContinuity = Sporadic;
  }

  Property CentralProcessDescription = "SpaceCCBuy = SpaceCCBuy_Thread
                                        SpaceCCBuy_Thread = SpaceCCBuy_login";

  Property CentralDataRecords : Set {TCentralDataRecord} = {
    [DatumID = "user";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
    [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
    [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
    [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
        ;],
    [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
    [DatumID = "cardNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
        Private;],
    [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
    [DatumID = "expDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited = Private
        ;],
    [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

  Property ComponentInOurControlDomain = No;
}

Component CompTWSAnalysisControl0 : CompTWSAnalysisControl = new CompTWSAnalysisControl
    extended with {
```

```
    Property ActiveAnalysisCentralDataStoreCorrect = true;
    Property ActiveAnalysisCommissionMismatch = true;
    Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
    Property ActiveAnalysisCommissionPartialMatch = true;
    Property ActiveAnalysisConcurrentCallsToThisPort = true;
    Property ActiveAnalysisMessageDataTypesMatch = true;
    Property ActiveAnalysisMessageExchangePatternsMatch = true;
    Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
    Property ActiveAnalysisMessageOverData = true;
    Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }

  Connector ConnTWS3 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWS4 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWS5 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWSStubborn0 : ConnTWSStubborn = new ConnTWSStubborn extended with { }
  Attachment SpaceCCBuy.login to ConnTWS3.role2;
  Attachment CPClient.setupConf to ConnTWS3.role1;
  Attachment SpaceCCBuy.checkCreditCard to ConnTWS4.role2;
  Attachment CPClient.PaymentCC to ConnTWS4.role1;
  Attachment CPClient.logout to ConnTWS5.role1;
  Attachment SpaceCCBuy.logout to ConnTWS5.role2;
  Attachment SpaceCCBuy.payByCC to ConnTWSStubborn0.role1;
}
```

E.2.4.2 SpaceCCBuy Alternate

```
import families/ws_enhanced_01.acme;
System AdditionalTestMultipleConnectionsSpaceCCBuyAlternate : ws_enhanced_01 = new ws_enhanced_01
      extended with {
  Component CPClient : CompTWSClient = new CompTWSClient extended with {
    Port setupConf : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
      Property MessagePattern = "SOLI
        CPClient_setupConf = CPClient_setupConf_sendReq -> CPClient_setupConf_p1
        CPClient_setupConf_p1 = CPClient_setupConf_p2 [] CPClient_setupConf_p3
        CPClient_setupConf_p2 = CPClient_setupConf_getRes -> CPClient_setupConf_OK
        CPClient_setupConf_p3 = CPClient_setupConf_getFault -> CPClient_setupConf_FAULT
        CPClient_setupConf_OK = CPClient_PaymentCC
        CPClient_setupConf_FAULT = CPClient_PaymentCC";

      Property Messages : TMessages = {
        [MessageId = "CPClient_setupConf_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_setupConf_getRes";MessageData = {
          [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_setupConf_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};
```

```

    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
    Property BindingSelfRemove = Yes;
    Property ChoiceGroup = "CarPark";
    Property DataContinuity = Sporadic;
    Property GroupChoiceMaker = Yes;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = Yes;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
  }

  Port PaymentCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
    Property MessagePattern = "SOLI
      CPClient_PaymentCC = CPClient_PaymentCC_sendReq -> CPClient_PaymentCC_p1
      CPClient_PaymentCC_p1 = CPClient_PaymentCC_p2 [] CPClient_PaymentCC_p3
      CPClient_PaymentCC_p2 = CPClient_PaymentCC_getRes -> CPClient_PaymentCC_OK
      CPClient_PaymentCC_p3 = CPClient_PaymentCC_getFault -> CPClient_PaymentCC_FAULT
      CPClient_PaymentCC_OK = CPClient_logout
      CPClient_PaymentCC_FAULT = CPClient_logout";

    Property Messages : TMessages = {
      [MessageId = "CPClient_PaymentCC_sendReq";MessageData = {
        [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
        [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private
            ;]};],
      [MessageId = "CPClient_PaymentCC_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_PaymentCC_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
    Property BindingSelfRemove = Yes;
    Property ChoiceGroup = "CarPark";
    Property DataContinuity = Sporadic;
    Property GroupChoiceMaker = No;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = Yes;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
         LateTimingFailures,HaltFailures,ErraticFailures};
```

```
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port logout : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
      Property MessagePattern = "SOLI
        CPClient_logout = CPClient_logout_sendReq -> CPClient_logout_p1
        CPClient_logout_p1 = CPClient_logout_p2 [] CPClient_logout_p3
        CPClient_logout_p2 = CPClient_logout_getRes -> CPClient_logout_OK
        CPClient_logout_p3 = CPClient_logout_getFault -> CPClient_logout_FAULT
        CPClient_logout_OK = CPClient_Thread
        CPClient_logout_FAULT = CPClient_Thread";

      Property Messages : TMessages = {
        [MessageId = "CPClient_logout_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_logout_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_logout_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property SendsFirstMessage = Yes;
      Property Reentrant = No;
      Property InOurControlDomain = Yes;
      Property GroupChoiceMaker = No;
      Property DataContinuity = Sporadic;
      Property ChoiceGroup = "CarPark";
      Property BindingSelfRemove = Yes;
      Property BindingSelfAdd = Yes;
      Property BindingOtherRemove = No;
      Property BindingOtherAdd = No;
      Property BindTime = Instantiation;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
      [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
      [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private;],
      [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
      [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited = Private;],
      [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
      [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited = Private;],
      [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

    Property CentralProcessDescription = "CPClient = CPClient_Thread
```

```
                                            CPClient_Thread = CPClient_setupConf";

    Property ComponentInOurControlDomain = Yes;
  }

  Component SpaceCCBuy : CompTWSService = new CompTWSService extended with {
    Port login : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        SpaceCCBuy_login = SpaceCCBuy_login_sendReq -> SpaceCCBuy_login_p1
        SpaceCCBuy_login_p1 = SpaceCCBuy_login_p2 [] SpaceCCBuy_login_p3
        SpaceCCBuy_login_p2 = SpaceCCBuy_login_getRes -> SpaceCCBuy_login_OK
        SpaceCCBuy_login_p3 = SpaceCCBuy_login_getFault -> SpaceCCBuy_login_FAULT
        SpaceCCBuy_login_OK = SpaceCCBuy_checkCreditCard
        SpaceCCBuy_login_FAULT = SpaceCCBuy_checkCreditCard";

      Property Messages : TMessages = {
        [MessageId = "SpaceCCBuy_login_sendReq";MessageData = {
          [DatumId = "user";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_login_getRes";MessageData = {
          [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_login_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = No;
      Property Reentrant = Yes;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/login"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
           LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
           LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
    }

    Port checkCreditCard : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        SpaceCCBuy_checkCreditCard = SpaceCCBuy_checkCreditCard_sendReq ->
             SpaceCCBuy_checkCreditCard_p1
        SpaceCCBuy_checkCreditCard_p1 = SpaceCCBuy_checkCreditCard_p2 []
             SpaceCCBuy_checkCreditCard_p3
        SpaceCCBuy_checkCreditCard_p2 = SpaceCCBuy_checkCreditCard_getRes ->
             SpaceCCBuy_checkCreditCard_OK
        SpaceCCBuy_checkCreditCard_p3 = SpaceCCBuy_checkCreditCard_getFault ->
             SpaceCCBuy_checkCreditCard_FAULT
        SpaceCCBuy_checkCreditCard_OK = SpaceCCBuy_payByCC
        SpaceCCBuy_checkCreditCard_FAULT = SpaceCCBuy_payByCC";
```

```

      Property Messages : TMessages = {
        [MessageId = "SpaceCCBuy_checkCreditCard_sendReq";MessageData = {
          [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "cardNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "expDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_checkCreditCard_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_checkCreditCard_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property InOurControlDomain = No;
      Property DataContinuity = Sporadic;
      Property BindingSelfRemove = No;
      Property BindingSelfAdd = No;
      Property BindingOtherRemove = Yes;
      Property BindingOtherAdd = Yes;
      Property BindTime = Instantiation;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/checkCreditCard"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
    }

    Port payByCC : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        SpaceCCBuy_payByCC = SpaceCCBuy_payByCC_sendReq -> SpaceCCBuy_payByCC_p1
        SpaceCCBuy_payByCC_p1 = SpaceCCBuy_payByCC_p2 [] SpaceCCBuy_payByCC_p3
        SpaceCCBuy_payByCC_p2 = SpaceCCBuy_payByCC_getRes -> SpaceCCBuy_payByCC_OK
        SpaceCCBuy_payByCC_p3 = SpaceCCBuy_payByCC_getFault -> SpaceCCBuy_payByCC_FAULT
        SpaceCCBuy_payByCC_OK = SpaceCCBuy_logout
        SpaceCCBuy_payByCC_FAULT = SpaceCCBuy_logout";

      Property Messages : TMessages = {
        [MessageId = "SpaceCCBuy_payByCC_sendReq";MessageData = {
          [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_payByCC_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_payByCC_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = No;
      Property Reentrant = No;
```

```
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/payByCC"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
    }

    Port logout : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        SpaceCCBuy_logout = SpaceCCBuy_logout_sendReq -> SpaceCCBuy_logout_p1
        SpaceCCBuy_logout_p1 = SpaceCCBuy_logout_p2 [] SpaceCCBuy_logout_p3
        SpaceCCBuy_logout_p2 = SpaceCCBuy_logout_getRes -> SpaceCCBuy_logout_OK
        SpaceCCBuy_logout_p3 = SpaceCCBuy_logout_getFault -> SpaceCCBuy_logout_FAULT
        SpaceCCBuy_logout_OK = SpaceCCBuy_Thread
        SpaceCCBuy_logout_FAULT = SpaceCCBuy_Thread";

      Property Messages : TMessages = {
        [MessageId = "SpaceCCBuy_logout_sendReq";MessageData = {
          [DatumId = "user";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_logout_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "SpaceCCBuy_logout_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property Reentrant = No;
      Property SendsFirstMessage = No;
      Property InOurControlDomain = No;
      Property BindingSelfRemove = No;
      Property BindingSelfAdd = No;
      Property BindingOtherRemove = Yes;
      Property BindingOtherAdd = Yes;
      Property BindTime = Instantiation;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.SpaceCCBuy/logout"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.SpaceCCBuy.com/WSDL"};
      Property DataContinuity = Sporadic;
    }

    Property CentralProcessDescription = "SpaceCCBuy = SpaceCCBuy_Thread
                                          SpaceCCBuy_Thread = SpaceCCBuy_login";

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "user";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
      [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
      [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
          ;],
```

```
      [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
      [DatumID = "cardNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited = Private;],
      [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
      [DatumID = "expDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited = Private;],
      [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

    Property ComponentInOurControlDomain = No;
  }

  Component CompTWSAnalysisControl0 : CompTWSAnalysisControl = new CompTWSAnalysisControl extended with {
    Property ActiveAnalysisCentralDataStoreCorrect = true;
    Property ActiveAnalysisCommissionMismatch = true;
    Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
    Property ActiveAnalysisCommissionPartialMatch = true;
    Property ActiveAnalysisConcurrentCallsToThisPort = true;
    Property ActiveAnalysisMessageDataTypesMatch = true;
    Property ActiveAnalysisMessageExchangePatternsMatch = true;
    Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
    Property ActiveAnalysisMessageOverData = true;
    Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }

 Connector ConnTWS3 : ConnTWS = new ConnTWS extended with { }
 Connector ConnTWS4 : ConnTWS = new ConnTWS extended with { }
 Connector ConnTWS5 : ConnTWS = new ConnTWS extended with { }
 Connector ConnTWSStubborn0 : ConnTWSStubborn = new ConnTWSStubborn extended with { }

 Attachment SpaceCCBuy.login to ConnTWS3.role2;
 Attachment CPClient.setupConf to ConnTWS3.role1;
 Attachment CPClient.PaymentCC to ConnTWS4.role1;
 Attachment CPClient.logout to ConnTWS5.role1;
 Attachment SpaceCCBuy.logout to ConnTWS5.role2;
 Attachment SpaceCCBuy.payByCC to ConnTWS4.role2;
 Attachment SpaceCCBuy.checkCreditCard to ConnTWSStubborn0.role1;
}
```

E.2.4.3 BookPayCC

```
import families/ws_enhanced_01.acme;
System AdditionalTestMultipleConnectionsBookPayCC : ws_enhanced_01 = new ws_enhanced_01 extended with {
  Component CPClient : CompTWSClient = new CompTWSClient extended with {
    Port setupConf : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
      Property MessagePattern = "SOLI
        CPClient_setupConf = CPClient_setupConf_sendReq -> CPClient_setupConf_p1
```

```
      CPClient_setupConf_p1 = CPClient_setupConf_p2 [] CPClient_setupConf_p3
      CPClient_setupConf_p2 = CPClient_setupConf_getRes -> CPClient_setupConf_OK
      CPClient_setupConf_p3 = CPClient_setupConf_getFault -> CPClient_setupConf_FAULT
      CPClient_setupConf_OK = CPClient_PaymentCC
      CPClient_setupConf_FAULT = CPClient_PaymentCC";

    Property Messages : TMessages = {
      [MessageId = "CPClient_setupConf_sendReq";MessageData = {
        [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_setupConf_getRes";MessageData = {
        [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_setupConf_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
    Property BindingSelfRemove = Yes;
    Property ChoiceGroup = "CarPark";
    Property DataContinuity = Sporadic;
    Property GroupChoiceMaker = Yes;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = Yes;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
  }

  Port PaymentCC : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
    Property MessagePattern = "SOLI
      CPClient_PaymentCC = CPClient_PaymentCC_sendReq -> CPClient_PaymentCC_p1
      CPClient_PaymentCC_p1 = CPClient_PaymentCC_p2 [] CPClient_PaymentCC_p3
      CPClient_PaymentCC_p2 = CPClient_PaymentCC_getRes -> CPClient_PaymentCC_OK
      CPClient_PaymentCC_p3 = CPClient_PaymentCC_getFault -> CPClient_PaymentCC_FAULT
      CPClient_PaymentCC_OK = CPClient_logout
      CPClient_PaymentCC_FAULT = CPClient_logout";

    Property Messages : TMessages = {
      [MessageId = "CPClient_PaymentCC_sendReq";MessageData = {
        [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
        [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
        [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private
            ;]};],
      [MessageId = "CPClient_PaymentCC_getRes";MessageData = {
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "CPClient_PaymentCC_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

```

```
      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property ChoiceGroup = "CarPark";
      Property DataContinuity = Sporadic;
      Property GroupChoiceMaker = No;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port logout : PortTWSClientUnicast = new PortTWSClientUnicast extended with {
      Property MessagePattern = "SOLI
        CPClient_logout = CPClient_logout_sendReq -> CPClient_logout_p1
        CPClient_logout_p1 = CPClient_logout_p2 [] CPClient_logout_p3
        CPClient_logout_p2 = CPClient_logout_getRes -> CPClient_logout_OK
        CPClient_logout_p3 = CPClient_logout_getFault -> CPClient_logout_FAULT
        CPClient_logout_OK = CPClient_Thread
        CPClient_logout_FAULT = CPClient_Thread";

      Property Messages : TMessages = {
        [MessageId = "CPClient_logout_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_logout_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "CPClient_logout_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property SendsFirstMessage = Yes;
      Property Reentrant = No;
      Property InOurControlDomain = Yes;
      Property GroupChoiceMaker = No;
      Property DataContinuity = Sporadic;
      Property ChoiceGroup = "CarPark";
      Property BindingSelfRemove = Yes;
      Property BindingSelfAdd = Yes;
      Property BindingOtherRemove = No;
      Property BindingOtherAdd = No;
      Property BindTime = Instantiation;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Property CentralDataRecords : Set {TCentralDataRecord} = {
```

```
      [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
      [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
      [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private;],
      [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
      [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited = Private;],
      [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
      [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited = Private;],
      [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

    Property CentralProcessDescription = "CPClient = CPClient_Thread
                                          CPClient_Thread = CPClient_setupConf";

    Property ComponentInOurControlDomain = Yes;
  }

  Component BookPayCC : CompTWSService = new CompTWSService extended with {
    Port setupConf : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        BookPayCC_setupConf = BookPayCC_setupConf_sendReq -> BookPayCC_setupConf_p1
        BookPayCC_setupConf_p1 = BookPayCC_setupConf_p2 [] BookPayCC_setupConf_p3
        BookPayCC_setupConf_p2 = BookPayCC_setupConf_getRes -> BookPayCC_setupConf_OK
        BookPayCC_setupConf_p3 = BookPayCC_setupConf_getFault -> BookPayCC_setupConf_FAULT
        BookPayCC_setupConf_OK = BookPayCC_PaymentCC
        BookPayCC_setupConf_FAULT = BookPayCC_PaymentCC";

      Property Messages : TMessages = {
        [MessageId = "BookPayCC_setupConf_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "password";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_setupConf_getRes";MessageData = {
          [DatumId = "success";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_setupConf_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Run;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = No;
      Property Reentrant = Yes;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/setupConf"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
```

```
    }

    Port PaymentCC : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        BookPayCC_PaymentCC = BookPayCC_PaymentCC_sendReq -> BookPayCC_PaymentCC_p1
        BookPayCC_PaymentCC_p1 = BookPayCC_PaymentCC_p2 [] BookPayCC_PaymentCC_p3
        BookPayCC_PaymentCC_p2 = BookPayCC_PaymentCC_getRes -> BookPayCC_PaymentCC_OK
        BookPayCC_PaymentCC_p3 = BookPayCC_PaymentCC_getFault -> BookPayCC_PaymentCC_FAULT
        BookPayCC_PaymentCC_OK = BookPayCC_logout
        BookPayCC_PaymentCC_FAULT = BookPayCC_logout";

      Property Messages : TMessages = {
        [MessageId = "BookPayCC_PaymentCC_sendReq";MessageData = {
          [DatumId = "owner";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "CCNumber";DatumRep = SOAP_String;DatumStateScopeExpected = Private;],
          [DatumId = "amount";DatumRep = SOAP_Float;DatumStateScopeExpected = Private;],
          [DatumId = "expirationDate";DatumRep = SOAP_Date;DatumStateScopeExpected = Private
               ;]};],
        [MessageId = "BookPayCC_PaymentCC_getRes";MessageData = {
          [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_PaymentCC_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property SendsFirstMessage = No;
      Property Reentrant = Yes;
      Property InOurControlDomain = No;
      Property DataContinuity = Sporadic;
      Property BindingSelfRemove = No;
      Property BindingSelfAdd = No;
      Property BindingOtherRemove = Yes;
      Property BindingOtherAdd = Yes;
      Property BindTime = Run;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/PaymentCC"};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
           LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
           LateTimingFailures,HaltFailures,ErraticFailures};
      Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
    }

    Port logout : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        BookPayCC_logout = BookPayCC_logout_sendReq -> BookPayCC_logout_p1
        BookPayCC_logout_p1 = BookPayCC_logout_p2 [] BookPayCC_logout_p3
        BookPayCC_logout_p2 = BookPayCC_logout_getRes -> BookPayCC_logout_OK
        BookPayCC_logout_p3 = BookPayCC_logout_getFault -> BookPayCC_logout_FAULT
        BookPayCC_logout_OK = BookPayCC_Thread
        BookPayCC_logout_FAULT = BookPayCC_Thread";

      Property Messages : TMessages = {
        [MessageId = "BookPayCC_logout_sendReq";MessageData = {
          [DatumId = "userName";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "BookPayCC_logout_getRes";MessageData = {
```

```
        [DatumId = "accepted";DatumRep = SOAP_Bool;DatumStateScopeExpected = Private;]};],
      [MessageId = "BookPayCC_logout_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property SendsFirstMessage = No;
    Property Reentrant = No;
    Property InOurControlDomain = No;
    Property DataContinuity = Sporadic;
    Property BindingSelfRemove = No;
    Property BindingSelfAdd = No;
    Property BindingOtherRemove = Yes;
    Property BindingOtherAdd = Yes;
    Property BindTime = Run;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property EndPointAddressList : TEndPointAddresses = {"www.BookPayCC/logout"};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property WsdlDocRefs : TWsdlDocs = {"www.BookPayCC.com/WSDL"};
  }

  Property CentralProcessDescription = "BookPayCC = BookPayCC_Thread
                                       BookPayCC_Thread = BookPayCC_setupConf";

  Property CentralDataRecords : Set {TCentralDataRecord} = {
    [DatumID = "userName";DatumSemantics = "USER:ID";DatumScopeExhibited = Private;],
    [DatumID = "password";DatumSemantics = "USER:KEY";DatumScopeExhibited = Private;],
    [DatumID = "success";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;],
    [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
        ;],
    [DatumID = "owner";DatumSemantics = "ACCOUNT:NAME";DatumScopeExhibited = Private;],
    [DatumID = "CCNumber";DatumSemantics = "ACCOUNT:CARD:REFERENCE";DatumScopeExhibited =
        Private;],
    [DatumID = "amount";DatumSemantics = "FINANCE:VALUE";DatumScopeExhibited = Private;],
    [DatumID = "expirationDate";DatumSemantics = "ACCOUNT:CARD:VALIDTO";DatumScopeExhibited =
        Private;],
    [DatumID = "accepted";DatumSemantics = "RESULT:FLAG";DatumScopeExhibited = Private;]};

  Property ComponentInOurControlDomain = No;
}

Component CompTWSAnalysisControl0 : CompTWSAnalysisControl = new CompTWSAnalysisControl
    extended with {
  Property ActiveAnalysisCentralDataStoreCorrect = true;
  Property ActiveAnalysisCommissionMismatch = true;
  Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
  Property ActiveAnalysisCommissionPartialMatch = true;
  Property ActiveAnalysisConcurrentCallsToThisPort = true;
  Property ActiveAnalysisMessageDataTypesMatch = true;
  Property ActiveAnalysisMessageExchangePatternsMatch = true;
  Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
  Property ActiveAnalysisMessageOverData = true;
  Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
```

```
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }

  Connector ConnTWS0 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS1 : ConnTWS = new ConnTWS extended with {  }
  Connector ConnTWS2 : ConnTWS = new ConnTWS extended with {  }

  Attachment BookPayCC.setupConf to ConnTWS0.role1;
  Attachment CPClient.setupConf to ConnTWS0.role2;
  Attachment BookPayCC.PaymentCC to ConnTWS1.role2;
  Attachment CPClient.PaymentCC to ConnTWS1.role1;
  Attachment CPClient.logout to ConnTWS2.role1;
  Attachment BookPayCC.logout to ConnTWS2.role2;
}
```

E.2.5 Multi Threading Check

```
import families/ws_enhanced_01.acme;
System AdditionalTestMultiThreadingSoli : ws_enhanced_01 = new ws_enhanced_01 extended with {
  Component Client : CompTWSClient = new CompTWSClient extended with {
    Port p1 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Client_p1 = Client_p1_sendReq -> Client_p1_p1
        Client_p1_p1 = Client_p1_p2 [] Client_p1_p3
        Client_p1_p2 = Client_p1_getRes -> Client_p1_OK
        Client_p1_p3 = Client_p1_getFault -> Client_p1_FAULT
        Client_p1_OK = Client_p2
        Client_p1_FAULT = Client_p2";

      Property Messages : TMessages = {
        [MessageId = "Client_p1_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p1_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p1_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = Yes;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
```

```
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
  }

  Port p2 : PortTWSClientSingle = new PortTWSClientSingle extended with {
    Property MessagePattern = "SOLI
      Client_p2 = Client_p2_sendReq -> Client_p2_p1
      Client_p2_p1 = Client_p2_p2 [] Client_p2_p3
      Client_p2_p2 = Client_p2_getRes -> Client_p2_OK
      Client_p2_p3 = Client_p2_getFault -> Client_p2_FAULT
      Client_p2_OK = Client_Multi_Thread
      Client_p2_FAULT = Client_Multi_Thread";

    Property Messages : TMessages = {
      [MessageId = "Client_p2_sendReq";MessageData = {
        [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Client_p2_getRes";MessageData = {
        [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Client_p2_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = No;
    Property BindingOtherRemove = No;
    Property BindingSelfAdd = Yes;
    Property BindingSelfRemove = Yes;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = Yes;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
  }

  Port p3 : PortTWSClientSingle = new PortTWSClientSingle extended with {
    Property MessagePattern = "SOLI
      Client_p3 = Client_p3_sendReq -> Client_p3_p1
      Client_p3_p1 = Client_p3_p2 [] Client_p3_p3
      Client_p3_p2 = Client_p3_getRes -> Client_p3_OK
      Client_p3_p3 = Client_p3_getFault -> Client_p3_FAULT
      Client_p3_OK = Client_p4
      Client_p3_FAULT = Client_p4";

    Property Messages : TMessages = {
      [MessageId = "Client_p3_sendReq";MessageData = {
        [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Client_p3_getRes";MessageData = {
        [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Client_p3_getFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

```

```
      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = Yes;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Port p4 : PortTWSClientSingle = new PortTWSClientSingle extended with {
      Property MessagePattern = "SOLI
        Client_p4 = Client_p4_sendReq -> Client_p4_p1
        Client_p4_p1 = Client_p4_p2 [] Client_p4_p3
        Client_p4_p2 = Client_p4_getRes -> Client_p4_OK
        Client_p4_p3 = Client_p4_getFault -> Client_p4_FAULT
        Client_p4_OK = Client_Single_Thread
        Client_p4_FAULT = Client_Single_Thread";

      Property Messages : TMessages = {
        [MessageId = "Client_p4_sendReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p4_getRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Client_p4_getFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = No;
      Property BindingOtherRemove = No;
      Property BindingSelfAdd = Yes;
      Property BindingSelfRemove = Yes;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = No;
      Property SendsFirstMessage = Yes;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
          LateTimingFailures,HaltFailures,ErraticFailures};
    }

    Property CentralDataRecords : Set {TCentralDataRecord} = {
      [DatumID = "sendData";DatumSemantics = "sendData";DatumScopeExhibited = Private;],
      [DatumID = "resultData";DatumSemantics = "resultData";DatumScopeExhibited = Private;],
      [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
          ;]};
```

```

    Property CentralProcessDescription = "Client = Client_Multi_Thread ||| Client_Multi_Thread ||| Client_Single_Thread
                                         Client_Multi_Thread = Client_p1
                                         Client_Single_Thread = Client_p3 ";

    Property ComponentInOurControlDomain = Yes;
  }

  Component Service : CompTWSService = new CompTWSService extended with {
    Port p1 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Service_p1 = Service_p1_getReq -> Service_p1_p1
        Service_p1_p1 = Service_p1_p2 [] Service_p1_p3
        Service_p1_p2 = Service_p1_sendRes -> Service_p1_OK
        Service_p1_p3 = Service_p1_sendFault -> Service_p1_FAULT
        Service_p1_OK = Service_p2
        Service_p1_FAULT = Service_p2";

      Property Messages : TMessages = {
        [MessageId = "Service_p1_getReq";MessageData = {
          [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Service_p1_sendRes";MessageData = {
          [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
        [MessageId = "Service_p1_sendFault";MessageData = {
          [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

      Property BindTime = Instantiation;
      Property BindingOtherAdd = Yes;
      Property BindingOtherRemove = Yes;
      Property BindingSelfAdd = No;
      Property BindingSelfRemove = No;
      Property DataContinuity = Sporadic;
      Property InOurControlDomain = Yes;
      Property Reentrant = Yes;
      Property SendsFirstMessage = No;
      Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
      Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
      Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,LateTimingFailures,HaltFailures,ErraticFailures};
      Property EndPointAddressList : TEndPointAddresses = {"www.Service.com/p1"};
      Property WsdlDocRefs : TWsdlDocs = {"www.Service.com/WSDL"};
    }

    Port p2 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
      Property MessagePattern = "REQR
        Service_p2 = Service_p2_getReq -> Service_p2_p1
        Service_p2_p1 = Service_p2_p2 [] Service_p2_p3
        Service_p2_p2 = Service_p2_sendRes -> Service_p2_OK
        Service_p2_p3 = Service_p2_sendFault -> Service_p2_FAULT
        Service_p2_OK = Service_Upper_Thread
        Service_p2_FAULT = Service_Upper_Thread";

```

```
    Property Messages : TMessages = {
      [MessageId = "Service_p2_getReq";MessageData = {
        [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p2_sendRes";MessageData = {
        [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p2_sendFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property EndPointAddressList : TEndPointAddresses = {"www.Service.com/p2"};
    Property WsdlDocRefs : TWsdlDocs = {"www.Service.com/WSDL"};
  }

  Port p3 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
    Property MessagePattern = "REQR
      Service_p3 = Service_p3_getReq -> Service_p3_p1
      Service_p3_p1 = Service_p3_p2 [] Service_p3_p3
      Service_p3_p2 = Service_p3_sendRes -> Service_p3_OK
      Service_p3_p3 = Service_p3_sendFault -> Service_p3_FAULT
      Service_p3_OK = Service_p4
      Service_p3_FAULT = Service_p4";

    Property Messages : TMessages = {
      [MessageId = "Service_p3_getReq";MessageData = {
        [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p3_sendRes";MessageData = {
        [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p3_sendFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = Yes;
    Property Reentrant = Yes;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
```

```
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property EndPointAddressList : TEndPointAddresses = {"www.Service.com/p3"};
    Property WsdlDocRefs : TWsdlDocs = {"www.Service.com/WSDL"};
  }

  Port p4 : PortTWSServiceSingle = new PortTWSServiceSingle extended with {
    Property MessagePattern = "REQR
      Service_p4 = Service_p4_getReq -> Service_p4_p1
      Service_p4_p1 = Service_p4_p2 [] Service_p4_p3
      Service_p4_p2 = Service_p4_sendRes -> Service_p4_OK
      Service_p4_p3 = Service_p4_sendFault -> Service_p4_FAULT
      Service_p4_OK = Service_Lower_Thread
      Service_p4_FAULT = Service_Lower_Thread";

    Property Messages : TMessages = {
      [MessageId = "Service_p4_getReq";MessageData = {
        [DatumId = "sendData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p4_sendRes";MessageData = {
        [DatumId = "resultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};],
      [MessageId = "Service_p4_sendFault";MessageData = {
        [DatumId = "FaultData";DatumRep = SOAP_String;DatumStateScopeExpected = Private;]};]};

    Property BindTime = Instantiation;
    Property BindingOtherAdd = Yes;
    Property BindingOtherRemove = Yes;
    Property BindingSelfAdd = No;
    Property BindingSelfRemove = No;
    Property DataContinuity = Sporadic;
    Property InOurControlDomain = Yes;
    Property Reentrant = No;
    Property SendsFirstMessage = No;
    Property EndPointList : TEndPoints = {[Transport = HTTP1_0;Encoding = SOAP1_1;]};
    Property FailureModesExpected : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property FailureModesExhibited : TFailureModes = {ContentFailures,EarlyTimingFailures,
        LateTimingFailures,HaltFailures,ErraticFailures};
    Property EndPointAddressList : TEndPointAddresses = {"www.Service.com/p4"};
    Property WsdlDocRefs : TWsdlDocs = {"www.Service.com/WSDL"};
  }

  Property CentralDataRecords : Set {TCentralDataRecord} = {
    [DatumID = "sendData";DatumSemantics = "sendData";DatumScopeExhibited = Private;],
    [DatumID = "resultData";DatumSemantics = "resultData";DatumScopeExhibited = Private;],
    [DatumID = "FaultData";DatumSemantics = "FAULT:DESCRIPTION";DatumScopeExhibited = Private
        ;]};

  Property CentralProcessDescription = "Service = Service_Upper_Thread ||| Service_Upper_Thread
       ||| Service_Lower_Thread ||| Service_Lower_Thread
                                      Service_Upper_Thread = Service_p1
                                      Service_Lower_Thread = Service_p3 ";

```

```
    Property ComponentInOurControlDomain = Yes;
  }

  Component AnalysisControl : CompTWSAnalysisControl = new CompTWSAnalysisControl extended with {
    Property ActiveAnalysisCentralDataStoreCorrect = true;
    Property ActiveAnalysisCommissionMismatch = true;
    Property ActiveAnalysisChoiceGroupsHaveChoiceMaker = true;
    Property ActiveAnalysisCommissionPartialMatch = true;
    Property ActiveAnalysisConcurrentCallsToThisPort = true;
    Property ActiveAnalysisMessageDataTypesMatch = true;
    Property ActiveAnalysisMessageExchangePatternsMatch = true;
    Property ActiveAnalysisMessageExchangePatternsPartiallyMatch = true;
    Property ActiveAnalysisMessageOverData = true;
    Property ActiveAnalysisMessagePatternAndMessageListConcur = true;
    Property ActiveAnalysisMessageUnderData1 = true;
    Property ActiveAnalysisMessageUnderData2 = true;
    Property ActiveAnalysisOmissionMismatch = true;
    Property ActiveAnalysisOmissionPartialMatch = true;
    Property ActiveAnalysisStateScopesMatch = true;
    Property outputPath = "";
  }

  Connector ConnTWS0 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWS1 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWS2 : ConnTWS = new ConnTWS extended with { }
  Connector ConnTWS3 : ConnTWS = new ConnTWS extended with { }

  Attachment Client.p1 to ConnTWS0.role1;
  Attachment Service.p1 to ConnTWS0.role2;
  Attachment Client.p3 to ConnTWS1.role1;
  Attachment Client.p2 to ConnTWS2.role2;
  Attachment Service.p4 to ConnTWS3.role1;
  Attachment Client.p4 to ConnTWS3.role2;
  Attachment Service.p2 to ConnTWS2.role1;
  Attachment Service.p3 to ConnTWS1.role2;
}
```

Appendix F

External Analysis Descriptions and Source Code

F.1 Class Group Outlines

There are 44 Java classes involved in the external analysis of this architectural style. They can be divided into seven groups, these will now be outlined to give an overview of the purpose of the classes.

F.1.1 External Analysis Main Classes

The first group includes those classes that Eclipse invokes when a particular external analysis is to be evaluated, there are 15 such classes in total. To reduce duplication of code, many of the classes do not themselves perform the analysis, but instead they use functions provided by classes in a shared library. While the names of the classes closely relate to the mismatches they target, they are all listed along with a brief description in Table F.1.

F.1.2 Message Pattern Comparison

The message pattern comparison class uses the ACME Interface classes to obtain data about the system. It then uses its own lookup table to determine if the message exchange patterns match or otherwise.

F.1.3 Message Comparison

Message comparison is carried out by four classes, Message Comparison, Message Mapping, Message Vector and Message Data Mapping. The message comparison starts by constructing a list of message mappings, mapping the IDs of the sent and received messages to allow them to be compared. This

Class(es)	*Description*
Commission mismatch Commission partial match Omission mismatch Omission partial match Concurrent calls to this port	These five classes check for commission, omission and concurrency mismatches. They all make use of the CSP modelling group of classes, invoking the CSP Model Builder with the choice of analysis and passing it the IDs of the required architecture elements.
Message data types match Message over data Message under data 1 Message under data 2 State scopes match	These classes look for mismatches relating to the semantics, data types and state scope assumptions declared for each datum in each message exchanged between a pair of ports. They all utilise a common message comparison class, described below.
Message exchange patterns match Message exchange patterns partially match	These classes compare the message exchanged patterns declared in each port. They do this by invoking a common message pattern comparison class, described below.
Central data store correct Message pattern and message list concur	These classes confirm a chain of data references. One checks that each message listed in the port message pattern CSP has a reference in the messages list, while the other checks that each datum in each message has is referenced in the central data store. They both perform their own analysis making use of the ACME Interface classes to obtain data.
Choice groups have choice maker	This class confirms that there is at least one port designated as a choice maker for each choice group on each component. It uses the ACME Interface to obtain data.

Table F.1: The main classes providing external analysis to the style grouped according by similar goals and the supporting classes they use.

is performed according to the data presented in Table 5.2 on page 82. Each message mapping is stored using a message vector instance to capture direction and IDs.

Each individual data pair in the mapped messages are then mapped onto each other for comparison, this mapping is recorded using the message data mapping class.

With the mappings in place the actual analysis required to check for data types and semantic loads of the messages is carried out using the data extraction utils to obtain properties from the ACME model. The one exception to this is the state scope assumptions which makes use of the ACME interface classes that were developed later.

F.1.4 Data Extraction Utils

The data extraction utils are a set of static methods that reduce the syntactic load involved in extracting data from the ACME Studio internal representation of a system.

F.1.5 CSP Modelling

The CSP modelling is managed by the CSP model builder class. This uses a number of other classes as follows:

Element CSP data stores the CSP descriptions of each element after they have been extracted from the system model and modified as needed;

CSP connector constructor stores the message IDs and their mappings to allow the connector process to be constructed;

CSP hiding set constructor stores the messages and events for each element to facilitate the hiding of these when required by the analysis being performed;

CSP memory constructor is used to construct the memory processes required when multiple connectors are attached to a single port;

CSP thread counter constructor generates the process to monitor the number of concurrent invocations of a port when checking for re entrance;

FDR results analyser parses the results returned by the FDR model checker and generates the results and output returned to the user.

These analysis classes make use of the ACME interface to obtain data about the system being modelled.

F.1.6 Acme Interface

The ACME interface class interrogates the system model presented by ACME studio and populates instances of the component, port and connector classes. This provides a more convenient means to obtain data about the system compared to the standard methods provided by ACME Studio for the analysis classes.

F.1.7 Exceptions

There are two exception classes defined:

Reportable exception is used where the problem should not occur, such as required properties not being present.

Acceptable exception allows analysis to terminate early when it is discovered that further investigation is not required. An example of this is when attempting to check the data types in message number four of a message exchange pattern. If the patterns of the two ports connected only share three messages then there is not a fourth message so the analysis uses the acceptable exception to exit early and force an analysis passed result to be returned.

F.1.8 Reporting

The results are reported using two classes:

Analysis result is a class used by all the external analysis classes. It contains a boolean indicating if an mismatch was found or not and also a string to hold a detailed description of the nature of a failure;

Reporter handles the writing of the detailed analysis output files if an analysis fails.

F.1.9 Data Types

There are three classes to representing data types:

Safe Boolean represents the safe boolean type used to make explicit the situations where the value is not defined;

Data Rep contains the representation of a datum and allows types to be compared for compatibility;

Data Semantics hold the semantics assigned to a datum and allows comparison for compatibility. As semantics are represented as strings in this work, compatibility is judged by string equality.

F.1.10 Support

The final classes included are those that provide general support.

Helper contains the methods supporting the output of debugging information and also contains the common methods used to write out CSP model files and to invoke the FDR model checker;

Look Up contains global static fields that are referenced by many of the classes for consistency;

Wait is used by some of the analysis to provide a small delay before evaluation commences, this was to make the ACME Studio interface more responsive;

Active analysis checker is used by all external analysis classes to determine whether they should perform their analysis or simply return a 'pass' result, again this was to improve performance when required. This class uses the `CompTWSAnalysisControl` element, Figure F.1 in the style to determine which analysis is active or not.

F.2 External analysis file outputs

The external analysis output a description of mismatches when found, there now follows an introduction to each output.

F.2.1 Commission Mismatch / Partial Match

These outputs inform the user of the event trace that leads to a commission event. An example of the format of output is as follows:

```
Broker attempted to send unexpected messages (commission events) in 1 traces.
Commision trace number 1

Broker_c3_sendReq
```

Here the analysis found a single trace leading to a commission and that trace contained a single message sent from the Broker component on port c3.

F.2.2 Omission Mismatch / Partial Match

These outputs inform the user of the trace observed by a component concluding in the expected message that is not received:

```
 ================================
```

```
Component Type CompTWSAnalysisControl = {
  Property ActiveAnalysisCommissionMismatch : boolean;
  Property ActiveAnalysisCommissionPartialMatch : boolean;
  Property ActiveAnalysisOmissionMismatch : boolean;
  Property ActiveAnalysisOmissionPartialMatch : boolean;
  Property ActiveAnalysisMessageExchangePatternsMatch : boolean;
  Property ActiveAnalysisMessageExchangePatternsPartiallyMatch : boolean;
  Property ActiveAnalysisConcurrentCallsToThisPort : boolean;
  Property ActiveAnalysisCentralDataStoreCorrect : boolean;
  Property ActiveAnalysisMessageDataTypesMatch : boolean;
  Property ActiveAnalysisMessageOverData : boolean;
  Property ActiveAnalysisMessageUnderData1 : boolean;
  Property ActiveAnalysisMessageUnderData2 : boolean;
  Property ActiveAnalysisStateScopesMatch : boolean;
  Property ActiveAnalysisMessagePatternAndMessageListConcur : boolean;
  Property ActiveAnalysisChoiceGroupsHaveChoiceMaker : boolean;
  Property outputPath : string;

  rule AnalysisCommissionMismatchActive =
       invariant ActiveAnalysisCommissionMismatch;
  rule AnalysisCommissionPartialMatchActive =
       invariant ActiveAnalysisCommissionPartialMatch;
  rule AnalysisOmissionMismatchActive =
       invariant ActiveAnalysisOmissionMismatch;
  rule AnalysisOmissionPartialMatchActive =
       invariant ActiveAnalysisOmissionPartialMatch;
  rule AnalysisMessageExchangePatternsMatchActive =
       invariant ActiveAnalysisMessageExchangePatternsMatch;
  rule AnalysisMessageExchangePatternsPartiallyMatchActive =
       invariant ActiveAnalysisMessageExchangePatternsPartiallyMatch;
  rule AnalysisConcurrentCallsToThisPortActive =
       invariant ActiveAnalysisConcurrentCallsToThisPort;
  rule AnalysisCentralDataStoreCorrectActive =
       invariant ActiveAnalysisCentralDataStoreCorrect;
  rule AnalysisMessageDataTypesMatchActive =
       invariant ActiveAnalysisMessageDataTypesMatch;
  rule AnalysisMessageOverDataActive =
       invariant ActiveAnalysisMessageOverData;
  rule AnalysisMessageUnderData1Active =
       invariant ActiveAnalysisMessageUnderData1;
  rule AnalysisMessageUnderData2Active =
       invariant ActiveAnalysisMessageUnderData2;
  rule AnalysisStateScopesMatchActive =
       invariant ActiveAnalysisStateScopesMatch;
  rule AnalysisMessagePatternAndMessageListConcurActive =
       invariant ActiveAnalysisMessagePatternAndMessageListConcur;
  rule AnalysisChoiceGroupsHaveChoiceMakerActive =
       invariant ActiveAnalysisChoiceGroupsHaveChoiceMaker;
}
```

Figure F.1: This describes the component type used to switch on and off specific externals analysis in a model.

```
[Broker_s1_getReq]
```

Here a single omission trace is shown for a component that never receives the message `Broker_s1_getReq`.

F.2.3 Concurrent Calls to this Port

This output simply confirms the result that two or more concurrent invocations of a non-reentrant port occurred:

```
This port experienced two or more simultaneous invocations
```

F.2.4 Message Data Types Match

This output informs the user of the IDs of the data with mismatching types along with the actual types sent and expected:

```
The data type ( SOAP_Int ) of Foo in the sent message is not compatible with the data
type ( SOAP_Float ) of Bar in the received message.
```

Here *Foo* has the data type *SOAP_Int* which is not directly compatible with the *SOAP_Float* expected for the *Bar* parameter.

F.2.5 Message Over Data

This output informs the user of which datum in the sent message are not expected by the recipient:

```
The following data was sent but is not expected:  owner
The following data was sent but is not expected:  CCNumber
The following data was sent but is not expected:  expirationDate
```

F.2.6 Message under Data 1

This informs the user that an expected item of data (*Foo)* is not in the sent message, but that an interrogation of the sending component's central data store indicates that data with the required semantics does exist (*Bar*):

```
There is no data in the message sent to match Foo, but it does appear to be
available in the sending component in datumID Bar
```

F.2.7 Message under Data 2

This informs the user that an expected item of data (*Foo)* is not in the sent message, and that an interrogation of the sending component's central data store indicates that it does not contain a suitable datum:

```
There is no data in the message sent to match Foo and it does not
appear to be available in the component
```

F.2.8 State Scopes Match

This analysis output reports each datum sent where the receiving component does not declare a compatible scope for that datum:

```
The datum Foo sent in message Login has expected data scope Private,
this is not compatible with the exhibited state Shared of the message datum
Bar it maps to
```

Here the sending component expects the receiving component to keep the *Foo* private, but the receiving component declares that it may share it.

F.2.9 Message Exchange Patterns Match

This analysis informs the user of mismatches caused by the choice of message exchange pattern, there are a number of output results.

If the patterns partially match:

```
These patterns partially match thanks to one or more of them being in our control
domain
```

If the patterns mismatch:

```
The patterns differ and neither port is in our control domain
```

If the patterns do not agree on the direction of the first message:

```
The patterns simply do not match due to message passing directions
```

F.2.10 Message Exchange Patterns Partially Match

This is the partner analysis to the previous example. It has two different output messages depending on the state of the mismatch.

If the patterns partially match then no output is produced. If the patterns mismatch:

```
The patterns differ and neither port is in our control domain
```

If the patterns do not agree on the direction of the first message:

```
The patterns simply do not match due to message passing directions
```

F.2.11 Central Data Store Correct

This analysis output informs the user if there are one or more data items in the messages that are not declared in the central data store:

```
The message Datum Foo exists in message CounterMessage in this port
CounterPort but does not exist in the central data store.
```

F.2.12 Message Pattern and Message List Concur

This analysis output informs the user if there is either a message declared in the CSP pattern that does not exist in the message list or vice versa.

If the message exists in the message pattern only:

```
the message Foo was found in the Message Exchange Pattern property but not
in the Messages
```

If the message exists in the messages list only:

```
the message Bar was found in the Messages property but not in the Message
Exchange Pattern
```

F.2.13 Choice Groups Have Choice Maker

This informs the user if there are any choice groups that have no choice maker:

```
The choice group Foo is without a choice maker
```

F.3 Message index numbers

In the style there are five rules that are repeated for each message in the message exchange pattern, these check the data types, semantics sent and expected, and the state scope expectations. The style labels the rules 1..4, however this does not help identify the message. Table F.2 presents a mapping showing the indexes and which message in the sequence they refer to from the following list:

message The initial request message in a sequence;

response a normal response to the first message;

	index	*ino*	*rio*	*reqr*	*ioo*
noti	1	message	message	message	message
roo	1	messsage	messsage	messsage	messsage
	2	N/A	fault	fault	fault
soli	1	messsage	messsage	messsage	messsage
	2	N/A	fault	response	response
	3	N/A	N/A	fault	fault
ooi	1	messsage	messsage	messsage	messsage
	2	N/A	fault	fault	fault
	3	N/A	N/A	response	response
	4	N/A	N/A	N/A	fault2

Table F.2: The message index numbers for each pairing of message exchange patterns.

fault1 a fault generated in response to the first message;

fault2 a fault generated in response to response to the first message.

F.4 Source Code

There now follows the complete source code for all the external analysis created in this work.

F.4.1 Acceptable Exception

```java
package uk.ac.ncl.cjg.ws_enhanced.common;

public class AcceptableException extends Exception {
  public AcceptableException()
  {
    super();
  }

  public AcceptableException(String message)
  {
    super(message);
  }

  public AcceptableException(String message, Throwable cause)
  {
    super(message, cause);
  }

  public AcceptableException(Throwable cause)
  {
    super(cause);
  }
}
```

F.4.2 Active Analysis Checker

```java
package uk.ac.ncl.cjg.ws_enhanced.common;

import org.acmestudio.acme.element.IAcmeElement;
import org.acmestudio.acme.element.IAcmeSystem;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.element.property.IAcmeProperty;
import org.acmestudio.acme.element.property.IAcmePropertyValue;
import org.acmestudio.acme.core.type.IAcmeBooleanValue;
import java.util.Set;
import java.util.Iterator;

import uk.ac.ncl.cjg.ws_enhanced.common.*;




public class ActiveAnalysisChecker {

  /**
   *   This method traverses the Acme system model till it finds an component of type
   *   CompTWSAnalysisControl, which it expects to find at the very highest level.  It checks
   *   the value of a property with the same name as the parameter ruleIDInTheStyle.  It returns
   *   the value of that property.  If no component of the right type is found, or if no
   *   property of the right name is found, the method will return true.
   *
   * @param ruleIDInTheStyle  The name of the rule
   * @param elementRuleIsIn the element from which the rule was invoked
   * @return       The value of the Active Analysis flag, if found, otherwise true
   */
  public static boolean CheckIfAnalysisIsActive(String ruleIDInTheStyle, IAcmeElement elementRuleIsIn) throws Exception{

    final String analysisControllerType = "CompTWSAnalysisControl";

    // move up the tree till we get the IAcmeSystem object
```

```java
    IAcmeElement theParent = elementRuleIsIn.getParent();
    IAcmeSystem theSystem = null;

    while(!(theParent instanceof IAcmeSystem))
    {
      theParent = theParent.getParent();
      if (theParent == null || !(theParent instanceof IAcmeElement)) return Boolean.TRUE;
    }
    theSystem = (IAcmeSystem)theParent;

    // get the list of all components in that system
    // move through the list till we find one of the correct type
    IAcmeComponent theAnalysisController = null;
    Set theComponents = theSystem.getComponents(); // maybe should parameterize the set here

    Iterator i = theComponents.iterator();
    while(i.hasNext())
    {
      IAcmeComponent thisComponent = (IAcmeComponent)i.next();
      if (thisComponent.declaresType(analysisControllerType))
      {
        theAnalysisController = thisComponent;
        break;
      }
    }

    if (theAnalysisController == null) throw new ReportableException("No analysis controller component found");

    // move through all properties of the component to find the one we are looking for
    // and return its value.

    IAcmeProperty analysisActiveProperty = theAnalysisController.getProperty(ruleIDInTheStyle);
    if (analysisActiveProperty == null) throw new ReportableException("Property controlling this analysis was not found");
    IAcmePropertyValue analysisActivePropertyValue = analysisActiveProperty.getValue();
    if (analysisActivePropertyValue instanceof IAcmeBooleanValue){
      if( ((IAcmeBooleanValue)analysisActivePropertyValue).getValue())
      {
        return Boolean.TRUE;
      }
      else
      {
        return Boolean.FALSE;
      }
    }

    throw new ReportableException("The property controlling this analysis did not have the type boolean");
  }

}
```

F.4.3 Acme Interface

```java
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
```

```java
import java.util.TreeSet;

import org.acmestudio.acme.core.type.IAcmeEnumValue;
import org.acmestudio.acme.core.type.IAcmeRecordField;
import org.acmestudio.acme.core.type.IAcmeRecordValue;
import org.acmestudio.acme.core.type.IAcmeSetValue;
import org.acmestudio.acme.core.type.IAcmeStringValue;
import org.acmestudio.acme.element.IAcmeAttachment;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.element.IAcmeElement;
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.element.IAcmeRole;
import org.acmestudio.acme.element.IAcmeSystem;
import org.acmestudio.acme.element.property.IAcmeProperty;

public class AcmeInterface {
  public Set elements;
  public Set conns;
  public Map<String, Port> ports;

  public static final int SAFE_BOOL_TRUE = 0;
  public static final int SAFE_BOOL_FALSE = 1;
  public static final int SAFE_BOOL_EMPTY = 2;

  public static final int DATUM_SCOPE_PRIVATE = 0;
  public static final int DATUM_SCOPE_PUBLIC = 1;
  public static final int DATUM_SCOPE_NO_PREFERENCE = 2;

  public AcmeInterface(IAcmeElement context) throws
      ReportableException {
    elements = new TreeSet();
    conns = new TreeSet();

    if (context instanceof IAcmeComponent) {
      buildAcmeModelFromComponent((IAcmeComponent) context);
    }

    if (context instanceof IAcmePort) {
      buildAcmeModelFromPort((IAcmePort) context);
    }

    if (context instanceof IAcmeConnector) {
      buildAcmeModelFromConnector((IAcmeConnector) context);
    }

  }

  public void buildAcmeModelFromPort(IAcmePort thePort)
      throws ReportableException {

    IAcmeComponent theComponent = (IAcmeComponent) thePort.
        getParent();
    IAcmeSystem theSystem = (IAcmeSystem) theComponent.
        getParent();
    buildModelFromRoot(theSystem);
  }

  public void buildAcmeModelFromComponent(IAcmeComponent
      theComponent)
      throws ReportableException {

    IAcmeSystem theSystem = (IAcmeSystem) theComponent.
        getParent();
    buildModelFromRoot(theSystem);
  }

  public void buildAcmeModelFromConnector(IAcmeConnector
      theConnector)
      throws ReportableException {
    IAcmeSystem theSystem = (IAcmeSystem) theConnector.
        getParent();
```

```java
    buildModelFromRoot(theSystem);
  }

  private void buildModelFromRoot(IAcmeSystem theSystem)
      throws ReportableException {
    Set allComponents = theSystem.getComponents();
    Set allConnectors = theSystem.getConnectors();
    Set tempPortSet = new TreeSet();
    ports = new TreeMap<String, Port>();
    Iterator compIt = allComponents.iterator();
    while (compIt.hasNext()) {
      IAcmeComponent thisComponent = (IAcmeComponent) compIt.
          next();
      Component tempComp = populateComponentFromAcme(
          thisComponent);

      if (tempComp != null) {
        elements.add(tempComp);

        Set compPorts = thisComponent.getPorts();

        Iterator portIt = compPorts.iterator();
        while (portIt.hasNext()) {
          IAcmePort thisPort = (IAcmePort) portIt.next();
          Port tempPort = populatePortFromAcme(thisPort);
          tempComp.addPort(tempPort);
          tempPortSet.add(tempPort);
          ports.put(thisPort.getQualifiedName(), tempPort);
        }
      }
    }

    Iterator connIt = allConnectors.iterator();
    while (connIt.hasNext()) {

      IAcmeConnector thisConnector = (IAcmeConnector) connIt.
          next();
      Connector tempConn = populateConnectorFromAcme(
          thisConnector,
          tempPortSet, theSystem);
      conns.add(tempConn);
    }
  }

  private Connector populateConnectorFromAcme(IAcmeConnector
      conn,
      Set thePorts, IAcmeSystem theSystem) throws
          ReportableException {
    // get type
    boolean connIsCooperative;
    boolean connIsStubborn;
    boolean connIsUnicast;
    if (conn.declaresType("ConnTWSCooperative")) {
      connIsCooperative = true;
    } else {
      connIsCooperative = false;
    }

    if (conn.declaresType("ConnTWSStubborn")) {
      connIsStubborn = true;
    } else {
      connIsStubborn = false;
    }

    // get ID
    String id = conn.getName();

    // get set of ports it is attached to
    String port1ID = null;
    String port2ID = null;
    Set roles = conn.getRoles();
```

```
    Iterator roleIt = roles.iterator();
    int index = 1;

    while (roleIt.hasNext()) {
      IAcmeRole thisRole = (IAcmeRole) roleIt.next();
      Set attachments = theSystem.getAttachments(thisRole);
      Iterator i = attachments.iterator();
      while (i.hasNext()) {
        IAcmeAttachment attach = (IAcmeAttachment) i.next();
        IAcmePort thisPort = attach.getPort();
        if (index == 1) {
          port1ID = thisPort.getQualifiedName();
        } else {
          port2ID = thisPort.getQualifiedName();
        }
        index++;
        break;
      }
    }

    Port port2 = null;
    Port port1 = ports.get(port1ID);
    if (!connIsCooperative && !connIsStubborn)
      port2 = ports.get(port2ID);

    // construct correct type
    if (port1 == null) {
      throw new ReportableException("Connnector " + id
          + " passed null for port1");
    }
    if (connIsCooperative) {
      return new Connector(id, port1, Connector.
          IS_COOPERATIVE_CONNECTOR);
    } else if (connIsStubborn) {
      return new Connector(id, port1, Connector.
          IS_STUBBORN_CONNECTOR);
    } else {
      if (port2 == null) {
        throw new ReportableException("Connnector " + id
            + " passed null for port2");
      }
      return new Connector(id, port1, port2);
    }
  }

  private Component populateComponentFromAcme(IAcmeComponent
      comp)
      throws ReportableException {

    // only process this component if it is not an analysis
        control one

    if (comp.declaresType("CompTWSAnalysisControl")) {
      return null;
    }

    Component thisComponent = new Component(comp.getName());

    // get centralProcessDescription
    IAcmeProperty cPD = comp.getProperty("
        CentralProcessDescription");
    if (cPD == null)
      throw new ReportableException("Component " +
          thisComponent.iD
          + " has no CentralProcessDescription");
    try {
      thisComponent.centralProcessDescription = ((
          IAcmeStringValue) (cPD
          .getValue())).getValue();
    } catch (Exception e) {
      throw new ReportableException(
          " the component "
```

```java
            + comp.getQualifiedName()
            + " has no value defined for its central process
                description");
    }

    // get centralDataRecords
    Set tempCDR = new TreeSet();
    IAcmeProperty cDR = comp.getProperty("CentralDataRecords");
    if (cDR == null)
      throw new ReportableException("Component " +
          thisComponent.iD
          + " has no CentralDataRecords");
    Set cDRSet = null;
    try {
      cDRSet = ((IAcmeSetValue) (cDR.getValue())).getValues();
    } catch (Exception e) {
      throw new ReportableException(" the component "
          + comp.getQualifiedName()
          + " has no value defined for its central data records
              ");
    }

    Iterator cDRSetIt = cDRSet.iterator();
    while (cDRSetIt.hasNext()) {

      IAcmeRecordValue thisRecord = (IAcmeRecordValue) cDRSetIt
          .next();

      IAcmeRecordField thisRecordField = thisRecord.getField("
          DatumID");
      Map centralDataRecord = new TreeMap();
      thisComponent.centralDataRecords.put(
          ((IAcmeStringValue) (thisRecordField.getValue()))
              .getValue(), centralDataRecord);

      thisRecordField = thisRecord.getField("DatumSemantics");
      centralDataRecord.put("DatumSemantics",
          ((IAcmeStringValue) (thisRecordField.getValue()))
              .getValue());

      thisRecordField = thisRecord.getField("
          DatumScopeExhibited");
      String theValue = ((IAcmeEnumValue) thisRecordField.
          getValue())
          .getValue();

      Integer tempInt = null;

      if (theValue.trim().equalsIgnoreCase("private")) {
        tempInt = new Integer(DATUM_SCOPE_PRIVATE);
      } else {
        tempInt = new Integer(DATUM_SCOPE_PUBLIC);
      }

      centralDataRecord.put("DatumScopeExhibited", tempInt);
    }

    // get component in our control domain
    IAcmeProperty iOCD = comp.getProperty("
        ComponentInOurControlDomain");
    if (iOCD == null)
      throw new ReportableException("Component " +
          thisComponent.iD
          + " has no value for ComponentInOurControlDomain");
    String iOCDVal = null;
    try {
      iOCDVal = ((IAcmeEnumValue) (iOCD.getValue())).getValue()
          ;
    } catch (Exception e) {
      throw new ReportableException(
          " the component "
              + comp.getQualifiedName()
```

```java
                + " has no value defined for its in our control
                    domain Property");
    }
    if (iOCDVal.trim().equalsIgnoreCase("YES")) {
      thisComponent.inOurControlDomain = true;
    } else if (iOCDVal.trim().equalsIgnoreCase("NO")) {
      thisComponent.inOurControlDomain = false;
    } else {
      throw new ReportableException("Component " +
          thisComponent.iD
          + " has no value for ComponentInOurControlDomain");
    }

    return thisComponent;
  }

  private Port populatePortFromAcme(IAcmePort port)
      throws ReportableException {
    Port thisPort = new Port(port.getName());

    // get messagePattern
    IAcmeProperty mP = port.getProperty("MessagePattern");
    if (mP == null)
      throw new ReportableException("Port " + thisPort.iD
          + " has no Message Pattern defined");

    try {
      thisPort.messagePattern = ((IAcmeStringValue) (mP.
          getValue()))
          .getValue();
    } catch (Exception e) {
      throw new ReportableException(" the port "
          + port.getQualifiedName()
          + " has no value defined for its messagePattern
              Property");
    }

    // get messages

    IAcmeProperty messages = port.getProperty("Messages");
    if (messages == null)
      throw new ReportableException("Port " + port.
          getQualifiedName()
          + " has no messages defined");

    IAcmeSetValue messagesSetValue = (IAcmeSetValue) messages.
        getValue();
    if (messagesSetValue == null)
      throw new ReportableException("Port " + port.
          getQualifiedName()
          + " has no values in the message property");
    Set messagesSet = messagesSetValue.getValues();
    Iterator messagesSetIt = messagesSet.iterator();

    while (messagesSetIt.hasNext()) {
      // get the message name and add a map to store the data
          items it
      // contains
      IAcmeRecordValue thisRecord = (IAcmeRecordValue)
          messagesSetIt
          .next();

      IAcmeRecordField messageIDRecord = thisRecord.getField("
          MessageId");
      if (messageIDRecord == null)
        throw new ReportableException("Port " + port.
            getQualifiedName()
            + " has a message with no ID");
      String messageID = ((IAcmeStringValue) messageIDRecord.
          getValue())
          .getValue();
      if (messageID == null)
```

```java
      throw new ReportableException("Port " + port.
          getQualifiedName()
          + " has a message with no ID");

    Map tempMessageMap = new TreeMap();
    thisPort.messages.put(messageID, tempMessageMap);

    // get the set of data items and add each to the data map
    IAcmeRecordField messageDataRecord = thisRecord
        .getField("MessageData");
    if (messageDataRecord == null)
      throw new ReportableException("Port " + port.
          getQualifiedName()
          + " has a message with no Data");

    IAcmeSetValue MessageDataSetValue = (IAcmeSetValue)
        messageDataRecord
        .getValue();
    if (MessageDataSetValue == null)
      throw new ReportableException("Port " + port.
          getQualifiedName()
          + " has a message with no Data");

    Set MessageDataSet = MessageDataSetValue.getValues();
    Iterator MessageDataSetIt = MessageDataSet.iterator();

    while (MessageDataSetIt.hasNext()) {

      // get the name of the data and then add a map to store
          its
      // properties
      IAcmeRecordValue thisDataRecord = (IAcmeRecordValue)
          MessageDataSetIt
          .next();

      List fieldsFound = thisDataRecord.getFields();
      String fieldsFoundList = "fields found list \n";
      Iterator ffi = fieldsFound.iterator();
      while (ffi.hasNext()) {
        fieldsFoundList += ((IAcmeRecordField) ffi.next())
            .getName();
      }

      IAcmeRecordField dataIDRecord = thisDataRecord
          .getField("DatumId");
      if (dataIDRecord == null)
        throw new ReportableException("Port "
            + port.getQualifiedName()
            + " has a message with a datum with no ID field"
            + "It actually contains \n" + fieldsFoundList);
      String dataID = ((IAcmeStringValue) dataIDRecord.
          getValue()).getValue();
      if (dataID == null)
        throw new ReportableException(
            "Port "
                + port.getQualifiedName()
                + " has a message with a datum with no ID
                    field value");

      Map tempDataMap = new TreeMap();
      tempMessageMap.put(dataID, tempDataMap);

      // get the data representation property and add it to
          the map
      IAcmeRecordField dataRepresentationRecord =
          thisDataRecord
          .getField("DatumRep");
      if (dataRepresentationRecord == null)
        throw new ReportableException(
            "Port "
                + port.getQualifiedName()
```

```
                + " has a message with a datum with no
                    representation");
        String dataRepresentation = ((IAcmeEnumValue)
            dataRepresentationRecord.getValue())
            .getValue();
        if (dataRepresentation == null)
          throw new ReportableException(
              "Port "
                  + port.getQualifiedName()
                  + " has a message with a datum with no
                      representation");

        tempDataMap.put("DatumRep", dataRepresentation);

        // get the state scope property and add it to the map
        IAcmeRecordField dataScopeRecord = thisDataRecord
            .getField("DatumStateScopeExpected");
        if (dataScopeRecord == null)
          throw new ReportableException(
              "Port "
                  + port.getQualifiedName()
                  + " has a message with a datum with no datum
                      scope stated");
        String dataStateScope = ((IAcmeEnumValue)
            dataScopeRecord.getValue())
            .getValue();
        if (dataStateScope == null)
          throw new ReportableException(
              "Port "
                  + port.getQualifiedName()
                  + " has a message with a datum with no datum
                      scope stated");

        tempDataMap.put("DatumStateScopeExpected",
            dataStateScope);
      }
    }

    // get reentrant
    IAcmeProperty r = port.getProperty("Reentrant");
    if (r == null)
      throw new ReportableException("Port " + port.
          getQualifiedName()
          + " is not explicit about whether it is reentrant");
    String rValue = null;

    try {
      rValue = ((IAcmeEnumValue) (r.getValue())).getValue();
    } catch (Exception e) {
      throw new ReportableException(" the port "
          + port.getQualifiedName()
          + " has no value defining if it is reentrant or not")
              ;
    }
    if (rValue.trim().equalsIgnoreCase("YES")) {
      thisPort.reentrant = true;
    } else if (rValue.trim().equalsIgnoreCase("NO")) {
      thisPort.reentrant = false;
    } else {
      throw new ReportableException("Port " + port.
          getQualifiedName()
          + " is not explicit about whether it is reentrant");
    }

    // get isUnicast - from type declared
    if (port.declaresType("PortTWSClientUnicast")
        || port.declaresType("PortTWSServiceUnicast")) {
      thisPort.isUnicast = true;
    } else {
      thisPort.isUnicast = false;
    }

```

```java
if (thisPort.isUnicast) {
  // get choice group - if required
  IAcmeProperty cG = port.getProperty("ChoiceGroup");
  if (cG == null)
    throw new ReportableException("Port " + port.getQualifiedName()
        + " has no choiceGroup defined");
  try {
    thisPort.choiceGroup = ((IAcmeStringValue) (cG.getValue()))
        .getValue();
  } catch (Exception e) {
    throw new ReportableException(" port :"
        + port.getQualifiedName()
        + " has no value defined for its ChoiceGroup property ");
  }
  // get choice group maker - if required

  IAcmeProperty gCM = port.getProperty("GroupChoiceMaker");
  if (gCM == null)
    throw new ReportableException("Port " + port.getQualifiedName()
        + " is not explicit about whether is a choice maker");
  String gCMValue = null;
  try {
    gCMValue = ((IAcmeEnumValue) (gCM.getValue())).getValue();
  } catch (Exception e) {
    throw new ReportableException(
        " the port "
            + port.getQualifiedName()
            + " has no value defined for its ChoiceGroupMaker property ");
  }
  if (gCMValue.trim().equalsIgnoreCase("YES")) {
    thisPort.choiceGroupMaker = true;
  } else if (gCMValue.trim().equalsIgnoreCase("NO")) {
    thisPort.choiceGroupMaker = false;
  } else {
    throw new ReportableException("Port " + port.getQualifiedName()
        + " is not explicit about whether is a choice maker");
  }
}
// get in our control domain

IAcmeProperty iOCD = port.getProperty("InOurControlDomain");
if (iOCD == null)
  throw new ReportableException(
      "Port "
          + thisPort.iD
          + " is not explicit about whether it is in our control domain");
String iOCDValue = ((IAcmeEnumValue) (iOCD.getValue())).getValue();
if (iOCDValue.trim().equalsIgnoreCase("YES")) {
  thisPort.inOurControlDomain = true;
} else if (iOCDValue.trim().equalsIgnoreCase("NO")) {
  thisPort.inOurControlDomain = false;
} else {
  throw new ReportableException(
      "Port "
          + port.getQualifiedName()
          + " is not explicit about whether it is in our control domain");
}

return thisPort;
```

```
  }

  public String toString() {
    String toReturn = "";

    toReturn += " the comps set has elements " + elements.size() + " \n";

    Iterator i1 = elements.iterator();
    while (i1.hasNext()) {
      Component thisComp = (Component) i1.next();
      toReturn += " COMP : " + thisComp.toString();
    }

    toReturn += " the conns set has elements " + conns.size() + " \n";

    Iterator i2 = conns.iterator();
    while (i2.hasNext()) {
      Connector thisConn = (Connector) i2.next();
      toReturn += " CONN : " + thisConn.toString();
    }

    return toReturn;
  }
}
```

```
package uk.ac.ncl.cjg.ws_enhanced.common;


/**
 *   A simple class to all the boolean result of the analysis and any
 *   string report to be passed back to the calling plugin from the code
 *   that performed the analysis.
 */
public class AnalysisResult {

  private boolean theResult;
  private String theReport;

  public AnalysisResult (boolean theResult, String theReport)
  {
    this.theResult = theResult;
    this.theReport = theReport;
  }

  public boolean getResult()
  {
    return theResult;
  }

  public String getReport()
  {
    return theReport;
  }

}
```

F.4.4 Central Data Store Correct

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeComponent;
```

```java
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.AcmeInterface;
import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.Component;
import uk.ac.ncl.cjg.ws_enhanced.common.Port;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class CentralDataStoreCorrect implements IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis

    Wait.delayAnalysis();

    // extract data types from analysis call, this should be passed
    // a single component
    String ruleID = "ActiveAnalysisCentralDataStoreCorrect";
    IAcmeComponent theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeComponent) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when getting the activity status of this analysis: \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
```

```java
            "There was a general Exception raised when getting the activity status of this analysis:\n",
            e);
      return Boolean.FALSE;
    }

    // perform the analysis
    try {

      // construct the acme interface and grab the required port from it
      String focusComponentID = theElement.getName();
      AcmeInterface ai = new AcmeInterface(theElement);

      // get the component from the interface and extract its central data
      // store
      // map keys, these are the datum IDs we need

      Component thisComponent = null;
      boolean componentFound = false;
      Iterator allElements = ai.elements.iterator();
      while (allElements.hasNext()) {
        thisComponent = (Component) allElements.next();
        if (thisComponent.iD.equalsIgnoreCase(focusComponentID)) {
          componentFound = true;
          break;
        }
      }
      if (!componentFound)
        throw new ReportableException(
            "The required component was not found in the model");

      Set componentDatumIDs = thisComponent.centralDataRecords.keySet();

      // extract the set of ports from this component and start a loop to
      // process each one

      Iterator allComponentPorts = thisComponent.ports.iterator();
      boolean allDatumInMessagesFoundInCentralDataStore = true;
      String reportDetails = "";

      while (allComponentPorts.hasNext()) {
        // get the datum id keys from with each message of this port,
        // compare each with the keys from the component central data
        // store, they should exist if the data store is correct.

        Port thisPort = (Port) allComponentPorts.next();

        Iterator thisPortMessages = thisPort.messages.keySet().iterator();
        while (thisPortMessages.hasNext()) {
          String thisMessage = (String) thisPortMessages.next();
          Map thisMessageData = (Map) thisPort.messages.get(thisMessage);
          Iterator thisMessageDataIt = thisMessageData.keySet().iterator();

          while (thisMessageDataIt.hasNext()) {
            String thisDatumID = (String) thisMessageDataIt.next();

```

```
          boolean thisMessageDatumFound = false;
          Iterator componentDatumIDsIt = componentDatumIDs
              .iterator();

          while (componentDatumIDsIt.hasNext()) {
            String thisComponentDatumID = (String)
                componentDatumIDsIt
                .next();
            if (thisComponentDatumID
                .equalsIgnoreCase(thisDatumID)) {
              thisMessageDatumFound = true;
              break;
            }
          }

          if (!thisMessageDatumFound) {
            allDatumInMessagesFoundInCentralDataStore = false
                ;
            reportDetails += "The message Datum "
                + thisDatumID
                + " exists in message "
                + thisMessage
                + " in this port "
                + thisPort.iD
                + " but does not exist in the central data
                    store. \n";
          }
        }
      }
    }

    theResult = new AnalysisResult(
        allDatumInMessagesFoundInCentralDataStore,
            reportDetails);

  } catch (ReportableException e) {
    Reporter.report(theElement, ruleID, e.getMessage());
    return Boolean.FALSE;
  } catch (Exception e) {
    Reporter
        .report(
            theElement,
            ruleID,
            "There was an Exception raised performing the
                analysis: \n",
            e);
    return Boolean.FALSE;
  }

  // report and return the results
  Reporter.report(theElement, ruleID, theResult.getReport());
  if (theResult.getResult() == true)
    return Boolean.TRUE;
  else
    return Boolean.FALSE;

 }

}
```

F.4.5 Choice Groups Have Choice Maker

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Stack;
import java.util.TreeMap;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeComponent;
```

```java
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.AcmeInterface;
import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.Component;
import uk.ac.ncl.cjg.ws_enhanced.common.Port;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class ChoiceGroupsHaveChoiceMaker implements
    IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {
    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis

    Wait.delayAnalysis();

    // extract data types from analysis call, this should be passed
    // a single component
    String ruleID = "ActiveAnalysisChoiceGroupsHaveChoiceMaker";
    IAcmeComponent theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeComponent) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID, "Some fo the required elements required "
          + "(the connector and both attached ports) were"
          + "not passed by acme to the analysis: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when getting the activity status of this analysis: \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
```

```
                ruleID,
                "There was a general Exception raised when
                    getting the activity status of this analysis:
                    \n",
                e);
        return Boolean.FALSE;
    }

    // perform the analysis
    try {
        AcmeInterface ai = new AcmeInterface(theElement);

        String thisComponentId = theElement.getName();
        Component thisComponent = null;

        Iterator allElements = ai.elements.iterator();
        while (allElements.hasNext()) {
            Component tempComp = (Component) allElements.next();
            if (tempComp.iD.equalsIgnoreCase(thisComponentId)) {
                thisComponent = tempComp;
                break;
            }
        }

        if (thisComponent == null)
            throw new Exception(
                "The component was not found in the AcmeInterface")
                    ;

        Iterator allPortsIt = thisComponent.ports.iterator();

        Map<String, Boolean> groups = new TreeMap<String, Boolean
            >();
        boolean unicastWithNoGroup = false;
        String reportDetails = "";

        while (allPortsIt.hasNext()) {
            Port thisPort = (Port) allPortsIt.next();

            if (thisPort.isUnicast) {
                if (thisPort.choiceGroup == null
                    || thisPort.choiceGroup.equalsIgnoreCase(""))
                    unicastWithNoGroup = true;
                else {
                    if (!groups.containsKey(thisPort.choiceGroup))
                        groups.put(thisPort.choiceGroup, new Boolean(
                            thisPort.choiceGroupMaker));
                    else if (thisPort.choiceGroupMaker)
                        groups.put(thisPort.choiceGroup, new Boolean(true
                            ));
                }
            }
        }

        boolean allGroupsHaveChoiceMaker = true;

        Iterator groupsIt = groups.keySet().iterator();
        while(groupsIt.hasNext())
        {
            String groupKey = (String)groupsIt.next();
            Boolean thisGroupHasChoiceMaker = groups.get(groupKey);
            if(!thisGroupHasChoiceMaker.booleanValue())
            {
                allGroupsHaveChoiceMaker = false;
                reportDetails += " The choice group " + groupKey + "
                    is without a choice maker \n";
            }
        }

        if(!allGroupsHaveChoiceMaker || unicastWithNoGroup)
            theResult = new AnalysisResult(false, reportDetails);
        else
```

```java
        theResult = new AnalysisResult(true, reportDetails);
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the
                  analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }
}
```

F.4.6 Commission Mismatch

```java
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.
    IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.
    AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.AcceptableException;
import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPConnectorConstructor
    ;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPHidingSetConstructor
    ;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPModelBuilder;
import uk.ac.ncl.cjg.ws_enhanced.common.FDRResultsAnalyzer;
import uk.ac.ncl.cjg.ws_enhanced.common.Helper;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class CommissionMismatch implements
    IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws
          AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something
        other than
    // external analysis

    Wait.delayAnalysis();



```

```java
    // extract data types from analysis call, this should be
        passed
    // a single component
    String ruleID = "ActiveAnalysisCommissionMismatch";
    IAcmeComponent theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeComponent) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n
              ", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID
          ,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when
                  getting the activity status of this analysis:
                   \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a general Exception raised when
                  getting the activity status of this analysis:
                   \n",
              e);
      return Boolean.FALSE;
    }

    // perform the analysis
    try {

      String outputPath = "/home/carl/analysisModel.csp";

      List fdrRawResults = new LinkedList<String>();
      String focusCompID = theElement.getName();
      int analysisChoice = CSPModelBuilder.ANALYSIS_DEADLOCK;


      ArrayList theModel = CSPModelBuilder.buildModel(
          analysisChoice,
          focusCompID, null, theElement);

      String theCSPModel = (String) theModel.get(0);
      CSPHidingSetConstructor hidCon = (CSPHidingSetConstructor
          ) theModel
          .get(1);
      CSPConnectorConstructor connCon = (
          CSPConnectorConstructor) theModel
          .get(2);

```

```java
      Helper.writeModelToFile(theCSPModel, outputPath);
      fdrRawResults = Helper.processCSPModel(outputPath, 100);

      FDRResultsAnalyzer ra = new FDRResultsAnalyzer(analysisChoice,
          hidCon, focusCompID, connCon);
      ra.submitDeadlockTraces(fdrRawResults);

      // ra.repoart results is true if the analysis failed, while the analysis
      // result expects a failed analysis to return false.
      if(ra.reportResult())
      {
        theResult = new AnalysisResult(false, ra.reportDetails());
      }
      else
      {
        theResult = new AnalysisResult(true, ra.reportDetails());
      }

      // theResult = MessageComparison.dataTypesMatch(port1, port2,
      // theMessageIndex);
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }

}
```

F.4.7 Commission Partial Match

```java
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPConnectorConstructor;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPHidingSetConstructor;
```

```java
import uk.ac.ncl.cjg.ws_enhanced.common.CSPModelBuilder;
import uk.ac.ncl.cjg.ws_enhanced.common.FDRResultsAnalyzer;
import uk.ac.ncl.cjg.ws_enhanced.common.Helper;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class CommissionPartialMatch implements IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis

    Wait.delayAnalysis();



    // extract data types from analysis call, this should be passed
    // a single component
    String ruleID = "ActiveAnalysisCommissionPartialMatch";
    IAcmeComponent theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeComponent) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when getting the activity status of this analysis: \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a general Exception raised when getting the activity status of this analysis: \n",
              e);
      return Boolean.FALSE;
    }
```

```java

    // perform the analysis
    try {

      String outputPath = "/home/carl/analysisModel.csp";

      List fdrRawResults = new LinkedList<String>();
      String focusCompID = theElement.getName();
      int analysisChoice = CSPModelBuilder.ANALYSIS_DEADLOCK_PARTIAL;


      ArrayList theModel = CSPModelBuilder.buildModel(analysisChoice,
          focusCompID, null, theElement);

      String theCSPModel = (String) theModel.get(0);
      CSPHidingSetConstructor hidCon = (CSPHidingSetConstructor) theModel
          .get(1);
      CSPConnectorConstructor connCon = (CSPConnectorConstructor) theModel
          .get(2);

      Helper.writeModelToFile(theCSPModel, outputPath);
      fdrRawResults = Helper.processCSPModel(outputPath, 100);

      FDRResultsAnalyzer ra = new FDRResultsAnalyzer(analysisChoice,
          hidCon, focusCompID, connCon);
      ra.submitDeadlockTraces(fdrRawResults);

      // ra.repoart results is true if the analysis failed, while the analysis
      // result expects a failed analysis to return false.
      if(ra.reportResult())
      {
        theResult = new AnalysisResult(false, ra.reportDetails());
      }
      else
      {
        theResult = new AnalysisResult(true, ra.reportDetails());
      }

      // theResult = MessageComparison.dataTypesMatch(port1, port2,
      // theMessageIndex);
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }

}
```

F.4.8 Component

```java
package uk.ac.ncl.cjg.ws_enhanced.common;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;

public class Component implements Comparable<Component>{
  public String iD;
  public String centralProcessDescription;
  public boolean inOurControlDomain;
  public Set ports;
  public Map centralDataRecords = new TreeMap();

  public Component (String iD)
  {
    this.iD = iD;
    ports = new TreeSet();
  }

  public void addPort(Port thePort)
  {
    ports.add(thePort);
    thePort.childOf = this;
  }

  public int compareTo(Component other)
  {
    return this.iD.compareTo(other.iD);
  }

  public String toString()
  {
    String toReturn ="";
    toReturn += "    ID " + iD + " \n";
    toReturn += "    Central process \n " +
        centralProcessDescription + " \n";
    toReturn += "    in our control domain \n " +
        inOurControlDomain + " \n";
    toReturn += "    has ports : \n";
    Iterator i1 = ports.iterator();
    while(i1.hasNext())
    {
      Port thisPort = (Port)i1.next();
      toReturn += thisPort.toString();
    }

    return toReturn;
  }

}
```

F.4.9 Concurrent Calls To This Port

```java
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.core.type.IAcmeEnumValue;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.element.property.IAcmeProperty;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.
    IExternalAnalysisExpressionNode;
```

```java
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPConnectorConstructor;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPHidingSetConstructor;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPModelBuilder;
import uk.ac.ncl.cjg.ws_enhanced.common.FDRResultsAnalyzer;
import uk.ac.ncl.cjg.ws_enhanced.common.Helper;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class ConcurrentCallsToThisPort implements
    IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis

    Wait.delayAnalysis();

    // extract data types from analysis call, this should be passed
    // a single component
    String ruleID = "ActiveAnalysisConcurrentCallsToThisPort";
    IAcmePort theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmePort) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when getting the activity status of this analysis: \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
```

```
                ruleID,
                "There was a general Exception raised when
                    getting the activity status of this analysis:
                    \n",
                e);
        return Boolean.FALSE;
    }

    // perform the analysis
    try {
        // first check for a reentrant port, these can not fail
            the analysis
        // so simply return a true

        IAcmeProperty reentrantProperty = theElement
            .getProperty("Reentrant");
        String reentrant = ((IAcmeEnumValue) reentrantProperty.
            getValue())
            .getValue();
        if (reentrant.equalsIgnoreCase("yes")) {
            // no need to proceed with the analysis, just return
                true;
            theResult = new AnalysisResult(true, "");
        } else {

            String outputPath = "/home/carl/analysisModel.csp";
            List fdrRawResults = new LinkedList<String>();

            String focusPortID = theElement.getName();
            String focusPortParentCompID = ((IAcmeComponent)
                theElement
                .getParent()).getName();

            int analysisChoice = CSPModelBuilder.
                ANALYSIS_THREAD_SPEC_REFINEMENT;

            ArrayList theModel = CSPModelBuilder.buildModel(
                analysisChoice,
                focusPortParentCompID, focusPortID, theElement);

            String theCSPModel = (String) theModel.get(0);
            CSPHidingSetConstructor hidCon = (
                CSPHidingSetConstructor) theModel
                .get(1);
            CSPConnectorConstructor connCon = (
                CSPConnectorConstructor) theModel
                .get(2);

            Helper.writeModelToFile(theCSPModel, outputPath);
            fdrRawResults = Helper.processCSPModel(outputPath, 100)
                ;

            FDRResultsAnalyzer ra = new FDRResultsAnalyzer(
                analysisChoice,
                hidCon, focusPortParentCompID, connCon);
            ra.submitRefinementTraces(fdrRawResults);

            // ra.repoart results is true if the analysis failed,
                while the
            // analysis
            // result expects a failed analysis to return false.
            if (ra.reportResult()) {
                theResult = new AnalysisResult(false, ra.
                    reportDetails());
            } else {
                theResult = new AnalysisResult(true, ra.reportDetails
                    ());
            }

            // theResult = MessageComparison.dataTypesMatch(port1,
                port2,
            // theMessageIndex);
```

```java
      }
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the
                  analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }

}
```

F.4.10 Connector

```java
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.Map;

public class Connector implements Comparable<Connector> {
  String iD;
  Port r1;
  Port r2;
  //public static final boolean IS_GOLDEN_CONNECTOR = true;
  //public static final boolean NOT_GOLDEN_CONNECTOR = false;
  public static final int IS_COOPERATIVE_CONNECTOR = 1;
  public static final int IS_COMMON_CONNECTOR = 2;
  public static final int IS_STUBBORN_CONNECTOR = 3;
  //boolean isGolden;
  int connType;

  public Connector(String iD, Port r1, Port r2) {
    this.iD = iD;
    this.r1 = r1;
    this.r2 = r2;
    //this.isGolden = NOT_GOLDEN_CONNECTOR;
    this.connType = IS_COMMON_CONNECTOR;

    r1.attachedTo.add(this);
    r2.attachedTo.add(this);
  }

  public Connector(String iD, Port r1, int connType) {
    this.iD = iD;
    this.r1 = r1;
    this.r2 = null;
    this.connType = connType;

    r1.attachedTo.add(this);
  }

  public int compareTo(Connector other) {
    return this.iD.compareTo(other.iD);
  }

  public String toString() {
    String toReturn = " a conn called ";

    toReturn += "    conn id : \n";
    toReturn += "" + iD + " \n";
```

```
    toReturn += "    port1 \n";
    toReturn += "" + r1.iD + " \n";

    toReturn += "    connector type (as an int, 1 = cooperative, 2 = common, 3 = stubborn) \n";
    toReturn += "" + connType + " \n";

    if (connType == IS_COMMON_CONNECTOR) {
      toReturn += "    port 2 \n";
      toReturn += "" + r2.iD + " \n";
    }
    return toReturn;
  }
}
```

F.4.11 CSP Connector Constructor

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.Iterator;
import java.util.Set;
import java.util.TreeSet;

public class CSPConnectorConstructor {

  private final String connectorProcessID = "CONN";
  private Set rowDataTuples = new TreeSet<ConnectorDataTuple>();

  public String getConnectorProcessID() {
    return connectorProcessID;
  }

  public void addSentMessage(String ConnectorID, String sentMsg,
      int mEPIndex, int connType, boolean portInOurControlDomain) {
    Helper.writeDebug(" trying to add a set msg " + sentMsg + " /n");

    addPartDataTuple(ConnectorID, sentMsg, null, mEPIndex, connType,
        portInOurControlDomain, false);
  }

  public void addReceivedMessage(String ConnectorID, String recvMsg,
      int mEPIndex, int connType, boolean portInOurControlDomain) {
    Helper.writeDebug(" trying to add a recv msg " + recvMsg + " /n");

    addPartDataTuple(ConnectorID, null, recvMsg, mEPIndex, connType,
        false, portInOurControlDomain);
  }

  public String getConnector() {

    Iterator rowElements = rowDataTuples.iterator();
    String conn = connectorProcessID + " = ";

    ConnectorDataTuple first = (ConnectorDataTuple) rowElements.next();
    if (first != null) {
      conn += first.getRow();
    }

    while (rowElements.hasNext()) {
      ConnectorDataTuple thisTuple = (ConnectorDataTuple) rowElements
```

```java
          .next();
      conn += "     [] " + thisTuple.getRow();
    }
    return conn;
  }

  public boolean isMessageUnderOurControl(String msgId)
      throws ReportableException {
    Iterator i = rowDataTuples.iterator();
    while (i.hasNext()) {
      ConnectorDataTuple thisOne = (ConnectorDataTuple) i.next
          ();

      if (thisOne.tupleContainsMessage(msgId)) {
        return thisOne.tupleContainsMessageUnderOurControl();
      }

    }
    throw new ReportableException(
        " Message not found when attempting to determine if is
            it under our control or not, problem with the data
            handling of the external analysis somewhere");
  }

  public boolean isReceivedMessageUnderOurControl(String msgId)
      throws ReportableException {
    Iterator i = rowDataTuples.iterator();
    while (i.hasNext()) {
      ConnectorDataTuple thisOne = (ConnectorDataTuple) i.next
          ();

      if (thisOne.tupleContainsMessage(msgId)) {
        Helper.writeDebug(" found a tuple that contains this
            message, recv under control = " + thisOne.
            tupleContainsReceivedMessageUnderOurControl());
        return thisOne.
            tupleContainsReceivedMessageUnderOurControl();
      }

    }
    throw new ReportableException(
        " Message not found when attempting to determine if is
            it under our control or not, problem with the data
            handling of the external analysis somewhere");
  }

  private void addPartDataTuple(String ConnectorID, String
      sentMsg,
      String recvMsg, int mEPIndex, int connType,
      boolean sentControl, boolean recvControl) {
    ConnectorDataTuple newTuple = new ConnectorDataTuple(
        ConnectorID,
        sentMsg, recvMsg, mEPIndex, connType, sentControl,
        recvControl);
    if (rowDataTuples.contains(newTuple)) {
      Iterator elements = rowDataTuples.iterator();
      boolean found = false;
      while (elements.hasNext() && !found) {
        ConnectorDataTuple thisTuple = (ConnectorDataTuple)
            elements
            .next();
        if (thisTuple.compareTo(newTuple) == 0) {
          found = true;
          if (sentMsg != null) {
            thisTuple.setSentMsg(sentMsg);
            thisTuple.setSentControl(sentControl);
          }
          if (recvMsg != null) {
            thisTuple.setRecvMsg(recvMsg);
            thisTuple.setRecvControl(recvControl);
          }
```

```
      }
    }
  } else {
    rowDataTuples.add(newTuple);
  }

}

private class ConnectorDataTuple implements Comparable<
    ConnectorDataTuple> {
  private String connectorID = null;
  private String sentMsg = null;
  private String recvMsg = null;
  private boolean sentUnderOurControl;
  private boolean recvUnderOurControl;
  private int mEPIndex = -1;
  private int connType;

  public ConnectorDataTuple(String connectorID, String
      sentMsg,
      String recvMsg, int mEPIndex, int connType,
      boolean sentUnderControl, boolean recvUnderControl) {
    this.connectorID = connectorID;
    this.sentMsg = sentMsg;
    this.recvMsg = recvMsg;
    this.mEPIndex = mEPIndex;
    this.connType = connType;
    this.sentUnderOurControl = sentUnderControl;
    this.recvUnderOurControl = recvUnderControl;
  }

  public String getConnectorID() {
    return connectorID;
  }

  public String getSentMsg() {
    return sentMsg;
  }

  public String getRecvMsg() {
    return recvMsg;
  }

  public int getMEPIndex() {
    return mEPIndex;
  }

  public void setSentMsg(String sentMsg) {
    this.sentMsg = sentMsg;
  }

  public void setSentControl(boolean ctrl) {
    this.sentUnderOurControl = ctrl;
  }

  public void setRecvMsg(String recvMsg) {
    this.recvMsg = recvMsg;
  }

  public void setRecvControl(boolean ctrl) {
    this.recvUnderOurControl = ctrl;
  }

  public boolean sentUnderControl() {
    return sentUnderOurControl;
  }

  public boolean recvUnderControl() {
    return recvUnderOurControl;
  }

  public boolean tupleContainsMessage(String msg) {
```

```
      Helper.writeDebug("sent message = " + sentMsg);
      Helper.writeDebug("Recv message = " + recvMsg);

      if(sentMsg != null && sentMsg.equals(msg))
        return true;

      if(recvMsg != null && recvMsg.equals(msg))
        return true;

      return false;
    }

    public boolean tupleContainsMessageUnderOurControl() {

      if (sentUnderOurControl || recvUnderOurControl)
        return true;
      else
        return false;

    }

    public boolean tupleContainsReceivedMessageUnderOurControl(){
      return recvUnderOurControl;
    }

    public String getRow() {
      String row = "";
      Helper.writeDebug(" the connector type value passed is " + connType);

      if (connType == Connector.IS_COMMON_CONNECTOR) {
        Helper.writeDebug(" common type processed");
        if (sentMsg == null) {
          row += "faux -> ";
        } else {
          row += sentMsg + " -> ";
        }

        if (recvMsg == null) {
          row += "faux -> " + connectorProcessID + "\n";
        } else {
          row += recvMsg + " -> " + connectorProcessID + " \n";
        }
      } else if(connType == Connector.IS_STUBBORN_CONNECTOR) {
        Helper.writeDebug(" stubborn type processed");
        if (sentMsg == null) {
          row += " faux -> " + recvMsg + " -> " + connectorProcessID + " \n";
        } else {
          row += sentMsg + " -> STOP \n";
        }
      } else {
        Helper.writeDebug(" coop type processed");
        if (sentMsg == null) {
          row += recvMsg + " -> " + connectorProcessID + " \n";
        } else {
          row += sentMsg + " -> " + connectorProcessID + " \n";
        }
      }
      return row;
    }

    public int compareTo(ConnectorDataTuple other) {
      String thisID = connectorID + mEPIndex;
      String otherID = other.connectorID + other.mEPIndex;
      return thisID.compareTo(otherID);
    }
  }
}
```

F.4.12 CSP Hiding Set Constructor

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class CSPHidingSetConstructor {

  private Map messages = new HashMap();
  private Map triggers = new HashMap();
  public static final int SENT_MESSAGE = 0;
  public static final int RECEIVED_MESSAGE = 1;

  public void addMessage(String compID, String message, int
      direction) {
    if (!messages.containsKey(compID)) {
      messages.put(compID,
          new TreeSet<ArrayList>(new msgDataComparator()));
    }
    Set valueSet = (TreeSet) messages.get(compID);
    ArrayList temp = new ArrayList();
    temp.add(null);
    temp.add(null);
    temp.set(0, message);
    temp.set(1, new Integer(direction));
    valueSet.add(temp);
  }

  public boolean compHasTriggers(String compID) {
    Set keySet = triggers.keySet();
    return keySet.contains(compID);
  }

  public boolean otherThanCompHasTriggers(String compID)
  {
    Set keySet = triggers.keySet();
    if(keySet.contains(compID))
      return (keySet.size()>1);
    else
      return (keySet.size()>0);
  }

  public boolean sysHasTriggers() {
    Set keySet = triggers.keySet();
    if (keySet.size() == 0)
      return false;
    else
      return true;
  }

  public void addTrigger(String compID, String trigger) {
    if (!triggers.containsKey(compID)) {
      triggers.put(compID, new TreeSet());
    }
    Set valueSet = (TreeSet) triggers.get(compID);
    valueSet.add(trigger);
  }

  public String getMessagesForComp(String compID) {
    String theList = "";
    Set compMsgs = (TreeSet) messages.get(compID);
    Iterator msgIt = compMsgs.iterator();

    ArrayList msgData = (ArrayList) msgIt.next();
    if (msgData != null) {
      theList = (String) msgData.get(0);
```

```
    }

    while (msgIt.hasNext()) {
      msgData = (ArrayList) msgIt.next();
      theList += ", " + (String) msgData.get(0);
    }
    return theList;
  }

  public String getTriggersForComp(String compID) {
    String theList = "";
    Set compTriggers = (TreeSet) triggers.get(compID);
    Iterator trgIt = compTriggers.iterator();

    String first = (String) trgIt.next();
    if (first != null) {
      theList = first;
    }

    String thisTrg = null;
    while (trgIt.hasNext()) {
      thisTrg = (String) trgIt.next();
      theList += ", " + thisTrg;
    }
    return theList;
  }

  public String getMesagesNotForComp(String compID) {
    Set compIDs = messages.keySet();
    Iterator compIDIt = compIDs.iterator();

    String theMessages = "";
    boolean first = true;
    String thisID;
    while (compIDIt.hasNext()) {
      thisID = (String) compIDIt.next();
      if (!thisID.equals(compID.trim())) {

        String compMsgs = getMessagesForComp(thisID);

        if (first) {
          first = false;
          theMessages += compMsgs;
        } else {
          theMessages += ", " + compMsgs;
        }
      }
    }
    return theMessages;
  }

  public Set getAllMessagesAndTriggers() {
    Set allEvents = new TreeSet();

    Set compIDs = messages.keySet();
    Iterator compIDIt = compIDs.iterator();
    String thisID;
    while (compIDIt.hasNext()) {
      thisID = (String) compIDIt.next();
      Set compMsgs = (TreeSet) messages.get(thisID);
      Iterator compMsgsIt = compMsgs.iterator();
      while (compMsgsIt.hasNext()) {
        ArrayList thisMsgData = (ArrayList) compMsgsIt.next();
        allEvents.add((String) thisMsgData.get(0));
      }
    }

    compIDs = triggers.keySet();
    compIDIt = compIDs.iterator();
    while (compIDIt.hasNext()) {
      thisID = (String) compIDIt.next();
      Set compTrgs = (TreeSet) triggers.get(thisID);
```

```
      Iterator compTrgsIt = compTrgs.iterator();
      while (compTrgsIt.hasNext()) {
        allEvents.add((String) compTrgsIt.next());
      }
    }

    return allEvents;
  }

  public String getTriggersNotForComp(String compID) {
    Set compIDs = triggers.keySet();
    Iterator compIDIt = compIDs.iterator();

    String theTriggers = "";
    boolean first = true;
    String thisID;
    while (compIDIt.hasNext()) {
      thisID = (String) compIDIt.next();
      if (!thisID.equals(compID.trim())) {
        Set compTrgs = (TreeSet) triggers.get(thisID);
        Iterator compTrgsIt = compTrgs.iterator();
        while (compTrgsIt.hasNext()) {
          if (!first) {
            theTriggers += ", ";
          } else {
            first = false;
          }
          theTriggers += (String) compTrgsIt.next();
        }
      }
    }
    return theTriggers;
  }

  public String getAllMessages() {
    String messageList = "";
    boolean first = true;
    Set compIDs = messages.keySet();
    Iterator compIDIt = compIDs.iterator();
    while (compIDIt.hasNext()) {
      String compID = (String) compIDIt.next();
      if (first) {
        messageList += "faux, " + getMessagesForComp(compID);
        first = false;
      } else {
        messageList += ", " + getMessagesForComp(compID);
      }
    }
    return messageList;
  }

  public String getAllTriggers() {
    String triggerList = "";
    boolean first = true;
    Set compIDs = triggers.keySet();
    Iterator compIDIt = compIDs.iterator();
    while (compIDIt.hasNext()) {
      String compID = (String) compIDIt.next();
      if (first) {
        triggerList += getTriggersForComp(compID);
        first = false;
      } else {
        triggerList += ", " + getTriggersForComp(compID);
      }
    }
    return triggerList;
  }

  public String getChannels() {
    String channelDec = "channel ";

    // add messages
```

```java
    channelDec += getAllMessages();

    if (sysHasTriggers()) {
      // add triggers
      channelDec += ", " + getAllTriggers() + " \n";
    }

    return channelDec;

  }

  public Set getSetMessagesForComp(String compID) {
    Set compMsgData = (TreeSet) messages.get(compID);
    Set theMessages = new TreeSet();
    Iterator i = compMsgData.iterator();

    while (i.hasNext()) {
      ArrayList thisData = (ArrayList) i.next();
      theMessages.add((String) thisData.get(0));
    }
    return theMessages;
  }

  public Set getSetMessagesDataForComp(String compID) {
    return (TreeSet) messages.get(compID);
  }

  public class msgDataComparator implements Comparator<
      ArrayList> {
    public int compare(ArrayList first, ArrayList second) {
      String firstName = (String) first.get(0);
      String secondName = (String) second.get(0);

      return firstName.compareTo(secondName);
    }
  }

}
```

F.4.13 CSP Memory Constructor

```java
package uk.ac.ncl.cjg.ws_enhanced.common;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class CSPMemoryConstructor {
  private Map memoryMaps = new HashMap();

  public void addChoiceMaker(String compID, String groupID,
      String message,
      String targetID) {
    addRecord(compID, groupID, message, targetID, true);
  }

  public void addChoiceFollower(String compID, String groupID,
      String message, String targetID) {
    addRecord(compID, groupID, message, targetID, false);
  }

  private void addRecord(String compID, String groupID, String
      message,
      String targetID, boolean choiceMaker) {
    Map thisComponent = null;
    Map thisGroup = null;
    ArrayList thisTarget = null;
    Set messages = null;

    // check for and add component if required
```

```java

    if (!memoryMaps.containsKey(compID)) {
      memoryMaps.put(compID, new HashMap());
    }
    thisComponent = (HashMap) memoryMaps.get(compID);

    // check for and add group if required
    if (!thisComponent.containsKey(groupID)) {
      thisComponent.put(groupID, new HashMap());
    }
    thisGroup = (HashMap) thisComponent.get(groupID);

    // check for and add target if required
    if (!thisGroup.containsKey(targetID)) {
      ArrayList temp = new ArrayList(2);
      temp.add(0, new TreeSet());
      temp.add(1, new TreeSet());
      thisGroup.put(targetID, temp);
    }
    thisTarget = (ArrayList) thisGroup.get(targetID);

    if (choiceMaker) {
      messages = (Set) thisTarget.get(0);
    } else {
      messages = (Set) thisTarget.get(1);
    }
    // just add the message to the set
    messages.add(message);
  }

  public String getComponentMemProcess(String compID) {

    String baseName = getComponentMemProcessID(compID);
    String componentMemoryProcesses = "";
    String componentMemoryProcessInterleave = null;
    Map initialProcesses = new HashMap();
    Map choicesMadeProcesses = new HashMap();

    // get value from compID

    Map compIDValue = (Map) memoryMaps.get(compID);

    // get set of groupIDs

    Set groupIDKeys = compIDValue.keySet();
    Iterator groupIDIt = groupIDKeys.iterator();

    while (groupIDIt.hasNext()) {
      String thisGroupID = (String) groupIDIt.next();
      Map groupIDValue = (Map) compIDValue.get(thisGroupID);

      // setup name and choice process string
      String choiceMakerProcessName = baseName + "_" +
          thisGroupID;
      String choiceMakerProcessUnnamed = "";

      // add choice maker to interleaving
      if (componentMemoryProcessInterleave == null) {
        componentMemoryProcessInterleave = baseName + " = "
            + choiceMakerProcessName;
      } else {
        componentMemoryProcessInterleave += " ||| "
            + choiceMakerProcessName + " \n";
      }

      // get set of target IDs
      Set targetIDKeys = groupIDValue.keySet();
      Iterator targetIDIt = targetIDKeys.iterator();
      boolean first = true;
      while (targetIDIt.hasNext()) {
        String thisTargetID = (String) targetIDIt.next();
        ArrayList targetIDValue = (ArrayList) groupIDValue
```

```java
            .get(thisTargetID);
        Set choiceMakers = (Set) targetIDValue.get(0);
        Iterator choiceMakersIt = choiceMakers.iterator();

        String choiceMakerTargetProcess =
            choiceMakerProcessName + "_"
            + thisTargetID;

        while (choiceMakersIt.hasNext()) {
          String theMessage = (String) choiceMakersIt.next();
          if (first) {
            choiceMakerProcessUnnamed += " = " + theMessage
                + " -> " + choiceMakerTargetProcess + "\n";
            first = false;
          } else {
            choiceMakerProcessUnnamed += "      [] " +
                theMessage
                + " -> " + choiceMakerTargetProcess + "\n";
          }
        }
      }

      componentMemoryProcesses += choiceMakerProcessName
          + choiceMakerProcessUnnamed + "\n \n";

      // now get the choice follower messages and create their
          processes

      // get set of target IDs
      targetIDKeys = groupIDValue.keySet();
      targetIDIt = targetIDKeys.iterator();

      while (targetIDIt.hasNext()) {
        String thisTargetID = (String) targetIDIt.next();
        ArrayList targetIDValue = (ArrayList) groupIDValue
            .get(thisTargetID);
        Set choiceFollowers = (Set) targetIDValue.get(1);
        Iterator choiceFollowersIt = choiceFollowers.iterator()
            ;

        String choiceFollowerTargetProcess =
            choiceMakerProcessName
            + "_" + thisTargetID;
        String choiceFollowerProcess = new String(
            choiceFollowerTargetProcess);
        choiceFollowerProcess += choiceMakerProcessUnnamed + "\
            n";

        // add target choice followers
        first = true;
        while (choiceFollowersIt.hasNext()) {
          String theMessage = (String) choiceFollowersIt.next()
              ;
          choiceFollowerProcess += "      [] " + theMessage + "
              -> "
              + choiceFollowerTargetProcess + "\n";

        }
        componentMemoryProcesses += choiceFollowerProcess + " \
            n";
      }
    }
    componentMemoryProcesses += " \n" +
        componentMemoryProcessInterleave;
    return componentMemoryProcesses;
  }

  private String getMapValue(String key) {
    return null;
  }

  public String getComponentMemProcessID(String compID) {
```

```
    return compID + "_ChoiceMemory";
  }

  public String synchProcessAndMemoryProcess(String processID,
      String compID) {

    boolean first = true;

    String toReturn = processID + " [| {| ";
    Map thisComp = (Map) memoryMaps.get(compID);
    Set groups = thisComp.keySet();
    Iterator groupIt = groups.iterator();
    while (groupIt.hasNext()) {
      String groupID = (String) groupIt.next();
      Map thisGroup = (Map) thisComp.get(groupID);
      Set targets = thisGroup.keySet();
      Iterator targetIt = targets.iterator();
      while (targetIt.hasNext()) {
        String targetID = (String) targetIt.next();
        ArrayList targetData = (ArrayList) thisGroup.get(
            targetID);

        Set choiceMakers = (Set) targetData.get(0);
        Iterator choiceMakerIt = choiceMakers.iterator();
        while (choiceMakerIt.hasNext()) {
          String thisMessage = (String) choiceMakerIt.next();
          if (!first) {
            toReturn += ", " + thisMessage;
          } else {
            toReturn += " " + thisMessage;
            first = false;
          }
        }

        Set choiceFollowers = (Set) targetData.get(1);
        Iterator choiceFollowerIt = choiceFollowers.iterator();
        while (choiceFollowerIt.hasNext()) {
          String thisMessage = (String) choiceFollowerIt.next()
              ;
          if (!first) {
            toReturn += ", " + thisMessage;
          } else {
            toReturn += " " + thisMessage;
            first = false;
          }
        }
      }
    }
    toReturn += " |} |] " + getComponentMemProcessID(compID);
    return toReturn;
  }

}
```


F.4.14 CSP Model Builder

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

import org.acmestudio.acme.element.IAcmeElement;

public class CSPModelBuilder {

  private static ArrayList modelData;

  public static final int ANALYSIS_DEADLOCK = 9;
```

```
public static final int ANALYSIS_DEADLOCK_PARTIAL = 10;
public static final int ANALYSIS_COMPONENT_REFINEMENT = 50;
public static final int ANALYSIS_COMPONENT_REFINEMENT_PARTIAL = 51;
public static final int ANALYSIS_DEADLOCK_OMISSION_SUPPORT = 57;
public static final int ANALYSIS_THREAD_SPEC_REFINEMENT = 1;

public static ArrayList buildModel(int selectedAnalysis,
    String focusComponentID, String focusPortID, IAcmeElement context)
    throws ReportableException {

  modelData = new ArrayList();

  // workaround as setting an initial size for the arrylist isnt working
  // :(
  modelData.add(null);
  modelData.add(null);
  modelData.add(null);
  modelData.add(null);

  ElementCSPData eleData = new ElementCSPData();
  modelData.set(0, new String()); // System
  modelData.set(1, new String()); // connector
  modelData.set(2, eleData); // elements
  modelData.set(3, new String()); // assertions

  CSPConnectorConstructor connCon = new CSPConnectorConstructor();
  CSPHidingSetConstructor hidCon = new CSPHidingSetConstructor();
  CSPMemoryConstructor memCon = new CSPMemoryConstructor();
  CSPThreadCounterConstructor threadCon = new CSPThreadCounterConstructor();

  // simplified acme interface to grab all required data
  AcmeInterface ai = new AcmeInterface(context);

  // process each component
  Set allComps = ai.elements;
  Iterator allCompsIt = allComps.iterator();
  while (allCompsIt.hasNext()) {
    // process each port on this component
    Component thisComp = (Component) allCompsIt.next();

    String compCSP = new String(thisComp.centralProcessDescription);
    String[] compCSPSplit = compCSP.split("\n");

    // add component CSP to data structure

    boolean thisCompHasUnicastPorts = false;

    Set allPorts = thisComp.ports;
    Iterator allPortsIt = allPorts.iterator();

    while (allPortsIt.hasNext()) {
      Port thisPort = (Port) allPortsIt.next();
      Helper.writeDebug(" CSPBuilder processing data from from port "
          + thisPort.iD);

      if (thisPort.isUnicast) {
        thisCompHasUnicastPorts = true;
      }

      String portCSP = new String(thisPort.messagePattern);
      String[] portCSPSplit = portCSP.split("\n");
      Map msgData = duplicateAndGetMessages(portCSPSplit,
          thisPort.attachedTo, thisPort, threadCon, memCon);
```

```

    Set messages = msgData.keySet();
    Iterator msgIterator = messages.iterator();

    while (msgIterator.hasNext()) {
      String msgName = ((String) msgIterator.next()).trim()
          ;
      ArrayList msgValue = (ArrayList) msgData.get(msgName)
          ;

      // add message to Connector

      String sentRecv = (String) msgValue.get(0);
      int mepIndex = ((Integer) msgValue.get(1)).intValue()
          ;
      String connID = (String) msgValue.get(2);
      int connType = ((Integer) msgValue.get(4)).intValue()
          ;

      if (sentRecv.equalsIgnoreCase("sent")) {
        connCon.addSentMessage(connID, msgName, mepIndex,
            connType, thisPort.inOurControlDomain);
        hidCon.addMessage(thisComp.iD, msgName,
            CSPHidingSetConstructor.SENT_MESSAGE);
      } else {
        connCon.addReceivedMessage(connID, msgName,
            mepIndex,
            connType, thisPort.inOurControlDomain);
        hidCon.addMessage(thisComp.iD, msgName,
            CSPHidingSetConstructor.RECEIVED_MESSAGE);
      }

      // add message to hiding sets

      // add faux triggers to counter and hiding set
      addTriggersToHidingSetAndCounter(thisPort.childOf.iD,
          thisPort.iD, portCSPSplit, threadCon, hidCon);

      // recombine the portCSPSplit and add to the
      // element data

      String newPortCSP = "";

      for (int index = 0; index < portCSPSplit.length;
          index++) {
        newPortCSP += portCSPSplit[index] + "\n";
      }
      eleData.addPort(thisComp.iD, thisPort.iD, newPortCSP)
          ;
    }
  }

  // add memory process to the component here
  String newCentralCSP = processCentralCSP(compCSPSplit,
      memCon
      .getComponentMemProcessID(thisComp.iD), thisComp.iD,
      memCon, thisCompHasUnicastPorts);

  eleData.addComponent(thisComp.iD, newCentralCSP);
}

// construct system model

String system = constructSys(eleData, hidCon, connCon);
modelData.set(0, system);

// add the connector
modelData.set(1, connCon.getConnector());

// This is where we will define the analysis

modelData.set(3, getAnalysisAssetions(selectedAnalysis,
```

```
        focusComponentID, focusPortID, memCon, hidCon,
            threadCon));

    String theModel = "";
    theModel += (String) modelData.get(0) + " \n";
    theModel += (String) modelData.get(1) + " \n";
    theModel += ((ElementCSPData) modelData.get(2)).
        getAllElements()
        + " \n";
    theModel += (String) modelData.get(3);

    ArrayList toReturn = new ArrayList();
    toReturn.add(null);
    toReturn.add(null);
    toReturn.add(null);

    toReturn.set(0, theModel);
    toReturn.set(1, hidCon);
    toReturn.set(2, connCon);
    return toReturn;
  }

  private static String getAnalysisAssetions(int
      selectedAnalysis,
      String focusComponentID, String focusPortID,
      CSPMemoryConstructor memCon, CSPHidingSetConstructor
          hidCon,
      CSPThreadCounterConstructor threadCon) {

    String analysisAssertions = "";

    if (selectedAnalysis == ANALYSIS_DEADLOCK
        || selectedAnalysis == ANALYSIS_DEADLOCK_PARTIAL) {
      // just assert the system is free from deadlock
      // system renamed so we can hide all triggers
      analysisAssertions += "ANALYSIS_SYSTEM = SYSTEM \\{| ";
      analysisAssertions += hidCon.getAllTriggers();
      analysisAssertions += " |} \n";
      analysisAssertions += "assert ANALYSIS_SYSTEM:[deadlock
          free[F]] \n";
    }

    if (selectedAnalysis == ANALYSIS_DEADLOCK_OMISSION_SUPPORT)
        {
      // just assert the system is refined by the specified
      // component, considering only those messages at its
      // interface

      analysisAssertions += " COMP_ONLY_SYSTEM = SYSTEM \\{| ";
      analysisAssertions += hidCon.getMesagesNotForComp(
          focusComponentID);

      if (hidCon.otherThanCompHasTriggers(focusComponentID))
        analysisAssertions += ", "
            + hidCon.getTriggersNotForComp(focusComponentID);

      if (hidCon.compHasTriggers(focusComponentID))
        analysisAssertions += ", "
            + hidCon.getTriggersForComp(focusComponentID);
      analysisAssertions += " |} \n";

      analysisAssertions += "assert COMP_ONLY_SYSTEM:[deadlock
          free[F]] \n";
    }

    if (selectedAnalysis == ANALYSIS_COMPONENT_REFINEMENT) {
      // just assert the system is refined by the specified
      // component, considering only those messages at its
      // interface

      analysisAssertions += "assert SYSTEM \\{| ";
```

```
      analysisAssertions += hidCon.getMesagesNotForComp(focusComponentID);
      if (hidCon.sysHasTriggers())
        analysisAssertions += ", " + hidCon.getAllTriggers();
      analysisAssertions += " |} [T= " + focusComponentID;
      if (hidCon.compHasTriggers(focusComponentID)) {
        analysisAssertions += " \\{| ";
        analysisAssertions += hidCon
            .getTriggersForComp(focusComponentID);
        analysisAssertions += " |} \n";
      }
    }

    if (selectedAnalysis == ANALYSIS_THREAD_SPEC_REFINEMENT) {
      // construct a new system synchronising on counter events
      // assert this is refined by the counter spec
      analysisAssertions += " channel Max \n \n";

      analysisAssertions += " SYSTEM_COUNTED = SYSTEM [| {| ";
      analysisAssertions += threadCon.getCounterTriggersForPort(
          focusComponentID, focusPortID);
      analysisAssertions += " |} |] " + threadCon.getCounterProcessName()
          + " \n \n";

      analysisAssertions += threadCon.getCounterProcess(focusComponentID,
          focusPortID)
          + "\n \n ";

      analysisAssertions += threadCon.getCounterSpec(focusComponentID,
          focusPortID, hidCon)
          + " \n \n";

      analysisAssertions += "assert " + threadCon.getCounterSpecName()
          + " [T= SYSTEM_COUNTED \n";

    }
    return analysisAssertions;
  }

  private static String constructSys(ElementCSPData eleData,
      CSPHidingSetConstructor hidCon, CSPConnectorConstructor connCon) {
    boolean first = true;
    String compsLine = "COMPS = ";
    Set compIDs = eleData.getCompIDs();
    Iterator compIDIt = compIDs.iterator();
    while (compIDIt.hasNext()) {
      String compID = (String) compIDIt.next();
      if (first) {
        compsLine += compID + " ";
        first = false;
      } else {
        compsLine += "||| " + compID;
      }
    }
    compsLine += " \n";

    String sysLine = "SYSTEM = COMPS [| {| ";
    sysLine += hidCon.getAllMessages();
    sysLine += " |} |] " + connCon.getConnectorProcessID();

    String channelDec = hidCon.getChannels();

    return channelDec + " \n " + compsLine + " \n " + sysLine + " \n";
  }

```

```
private static void addTriggersToHidingSetAndCounter(String compID,
    String portID, String[] thePattern,
    CSPThreadCounterConstructor threadCon,
    CSPHidingSetConstructor hidCon) {
  if (!thePattern[0].trim().equalsIgnoreCase("soli")
     && !thePattern[0].trim().equalsIgnoreCase("reqr")) {
    addNonMessageTrigger(compID, portID, thePattern[2],
        CSPThreadCounterConstructor.DEC_TRIGGER, threadCon, hidCon);
  }
}

private static void addNonMessageTrigger(String compID, String portID,
    String theLine, int triggerType,
    CSPThreadCounterConstructor threadCon,
    CSPHidingSetConstructor hidCon) {

  String[] theLineSplit = theLine.split(" ");
  String theTrigger = theLineSplit[2];

  if (triggerType == CSPThreadCounterConstructor.INC_TRIGGER) {
    threadCon.addIncEvent(compID, portID, theTrigger);
  } else {
    threadCon.addDecEvent(compID, portID, theTrigger);
  }

  hidCon.addTrigger(compID, theTrigger);
}

private static Map duplicateAndGetMessages(String[] thePattern,
    Set theConnectors, Port thisPort,
    CSPThreadCounterConstructor threadCon, CSPMemoryConstructor memCon) {

  Map msgData = new HashMap();

  if (thePattern[0].trim().equalsIgnoreCase("noti")) {
    thePattern[LookUP.CSP_INDEX_NOTI_SENDREQ] = duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_NOTI_SENDREQ], theConnectors,
        thisPort, true, LookUP.MESSAGE_INDEX_REQUEST, msgData,
        CSPThreadCounterConstructor.INC_TRIGGER, threadCon, memCon);
  }

  if (thePattern[0].trim().equalsIgnoreCase("roo")) {
    thePattern[LookUP.CSP_INDEX_ROO_SENDREQ] = duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_ROO_SENDREQ], theConnectors,
        thisPort, true, LookUP.MESSAGE_INDEX_REQUEST, msgData,
        CSPThreadCounterConstructor.INC_TRIGGER, threadCon, memCon);
    thePattern[LookUP.CSP_INDEX_ROO_GETFAULT] = duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_ROO_GETFAULT], theConnectors,
        thisPort, false, LookUP.MESSAGE_INDEX_FAULT, msgData,
        CSPThreadCounterConstructor.NOT_TRIGGER, threadCon, memCon);
  }

  if (thePattern[0].trim().equalsIgnoreCase("soli")) {
```

```
      thePattern[LookUP.CSP_INDEX_SOLI_SENDREQ] =
          duplicateMsgsOnLine(
          thePattern[LookUP.CSP_INDEX_SOLI_SENDREQ],
              theConnectors,
          thisPort, true, LookUP.MESSAGE_INDEX_REQUEST, msgData
              ,
          CSPThreadCounterConstructor.INC_TRIGGER, threadCon,
              memCon);
      thePattern[LookUP.CSP_INDEX_SOLI_GETRES] =
          duplicateMsgsOnLine(
          thePattern[LookUP.CSP_INDEX_SOLI_GETRES],
              theConnectors,
          thisPort, false, LookUP.MESSAGE_INDEX_RESPONSE,
              msgData,
          CSPThreadCounterConstructor.DEC_TRIGGER, threadCon,
              memCon);
      thePattern[LookUP.CSP_INDEX_SOLI_GETFAULT] =
          duplicateMsgsOnLine(
          thePattern[LookUP.CSP_INDEX_SOLI_GETFAULT],
              theConnectors,
          thisPort, false, LookUP.MESSAGE_INDEX_FAULT, msgData,
          CSPThreadCounterConstructor.DEC_TRIGGER, threadCon,
              memCon);
    }

    if (thePattern[0].trim().equalsIgnoreCase("ooi")) {
      thePattern[LookUP.CSP_INDEX_OOI_SENDREQ] =
          duplicateMsgsOnLine(
          thePattern[LookUP.CSP_INDEX_OOI_SENDREQ],
              theConnectors,
          thisPort, true, LookUP.MESSAGE_INDEX_REQUEST, msgData
              ,
          CSPThreadCounterConstructor.INC_TRIGGER, threadCon,
              memCon);
      thePattern[LookUP.CSP_INDEX_OOI_GETFAULT] =
          duplicateMsgsOnLine(
          thePattern[LookUP.CSP_INDEX_OOI_GETFAULT],
              theConnectors,
          thisPort, false, LookUP.MESSAGE_INDEX_FAULT, msgData,
          CSPThreadCounterConstructor.NOT_TRIGGER, threadCon,
              memCon);
      thePattern[LookUP.CSP_INDEX_OOI_GETRES] =
          duplicateMsgsOnLine(
          thePattern[LookUP.CSP_INDEX_OOI_GETRES],
              theConnectors,
          thisPort, false, LookUP.MESSAGE_INDEX_RESPONSE,
              msgData,
          CSPThreadCounterConstructor.NOT_TRIGGER, threadCon,
              memCon);
      thePattern[LookUP.CSP_INDEX_OOI_SENDFAULT2] =
          duplicateMsgsOnLine(
          thePattern[LookUP.CSP_INDEX_OOI_SENDFAULT2],
              theConnectors,
          thisPort, true, LookUP.MESSAGE_INDEX_FAULT2, msgData,
          CSPThreadCounterConstructor.NOT_TRIGGER, threadCon,
              memCon);
    }

    if (thePattern[0].trim().equalsIgnoreCase("ino")) {
      thePattern[LookUP.CSP_INDEX_INO_GETREQ] =
          duplicateMsgsOnLine(
          thePattern[LookUP.CSP_INDEX_INO_GETREQ],
              theConnectors,
          thisPort, false, LookUP.MESSAGE_INDEX_REQUEST,
              msgData,
          CSPThreadCounterConstructor.INC_TRIGGER, threadCon,
              memCon);
    }

    if (thePattern[0].trim().equalsIgnoreCase("rio")) {
      thePattern[LookUP.CSP_INDEX_RIO_GETREQ] =
          duplicateMsgsOnLine(
```

```
        thePattern[LookUP.CSP_INDEX_RIO_GETREQ],
            theConnectors,
        thisPort, false, LookUP.MESSAGE_INDEX_REQUEST,
            msgData,
        CSPThreadCounterConstructor.INC_TRIGGER, threadCon,
            memCon);
    thePattern[LookUP.CSP_INDEX_RIO_SENDFAULT] =
        duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_RIO_SENDFAULT],
            theConnectors,
        thisPort, true, LookUP.MESSAGE_INDEX_FAULT, msgData,
        CSPThreadCounterConstructor.NOT_TRIGGER, threadCon,
            memCon);
  }

  if (thePattern[0].trim().equalsIgnoreCase("reqr")) {
    thePattern[LookUP.CSP_INDEX_REQR_GETREQ] =
        duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_REQR_GETREQ],
            theConnectors,
        thisPort, false, LookUP.MESSAGE_INDEX_REQUEST,
            msgData,
        CSPThreadCounterConstructor.INC_TRIGGER, threadCon,
            memCon);
    thePattern[LookUP.CSP_INDEX_REQR_SENDRES] =
        duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_REQR_SENDRES],
            theConnectors,
        thisPort, true, LookUP.MESSAGE_INDEX_RESPONSE,
            msgData,
        CSPThreadCounterConstructor.DEC_TRIGGER, threadCon,
            memCon);
    thePattern[LookUP.CSP_INDEX_REQR_SENDFAULT] =
        duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_REQR_SENDFAULT],
            theConnectors,
        thisPort, true, LookUP.MESSAGE_INDEX_FAULT, msgData,
        CSPThreadCounterConstructor.DEC_TRIGGER, threadCon,
            memCon);
  }

  if (thePattern[0].trim().equalsIgnoreCase("ioo")) {
    thePattern[LookUP.CSP_INDEX_IOO_GETREQ] =
        duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_IOO_GETREQ],
            theConnectors,
        thisPort, false, LookUP.MESSAGE_INDEX_REQUEST,
            msgData,
        CSPThreadCounterConstructor.INC_TRIGGER, threadCon,
            memCon);
    thePattern[LookUP.CSP_INDEX_IOO_SENDFAULT] =
        duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_IOO_SENDFAULT],
            theConnectors,
        thisPort, true, LookUP.MESSAGE_INDEX_FAULT, msgData,
        CSPThreadCounterConstructor.NOT_TRIGGER, threadCon,
            memCon);
    thePattern[LookUP.CSP_INDEX_IOO_SENDRES] =
        duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_IOO_SENDRES],
            theConnectors,
        thisPort, true, LookUP.MESSAGE_INDEX_RESPONSE,
            msgData,
        CSPThreadCounterConstructor.NOT_TRIGGER, threadCon,
            memCon);
    thePattern[LookUP.CSP_INDEX_IOO_GETFAULT2] =
        duplicateMsgsOnLine(
        thePattern[LookUP.CSP_INDEX_IOO_GETFAULT2],
            theConnectors,
        thisPort, false, LookUP.MESSAGE_INDEX_FAULT2, msgData
            ,
```

```
            CSPThreadCounterConstructor.NOT_TRIGGER, threadCon,
                memCon);
    }

    return msgData;
  }

  private static String duplicateMsgsOnLine(String theLine,
      Set theConnectors, Port thisPort, boolean sent, int
          mepIndex,
      Map msgData, int triggerValue,
      CSPThreadCounterConstructor threadCon,
          CSPMemoryConstructor memCon) {

    // there is something going on with the leading spaces on
        the first line
    // so trying a trim to get rid of them
    String temp = theLine.trim();
    // Helper.writeDebug(" the line trimmed... agian " + temp);
    String[] lineSplit = temp.split(" ");

    String newLine = lineSplit[0] + " " + lineSplit[1];
    String message = lineSplit[2];
    String commonEnd = lineSplit[3] + " " + lineSplit[4];
    String otherCompID = null;

    Iterator i = theConnectors.iterator();
    boolean first = true;

    while (i.hasNext()) {
      Connector thisCon = (Connector) i.next();

      String messageID = "";

      Helper.writeDebug(" about to look at " + thisCon.iD
          + " which has conntype " + thisCon.connType);

      if (thisCon.connType == Connector.IS_COMMON_CONNECTOR) {
        // Helper.writeDebug(" A normal connector");
        Port r1 = thisCon.r1;
        Port r2 = thisCon.r2;
        if (r1 == thisPort) {
          otherCompID = r2.childOf.iD;
        } else {
          otherCompID = r1.childOf.iD;
        }

        messageID = message + "_" + otherCompID;

        if (first) {
          newLine += " " + messageID + " " + commonEnd;
          first = false;
        } else {
          newLine += " [] " + messageID + " " + commonEnd;
        }

        ArrayList msgDataValue = genMsgData(sent, mepIndex,
            thisCon.iD,
            otherCompID, Connector.IS_COMMON_CONNECTOR);

        msgData.put(message + "_" + otherCompID, msgDataValue);
      } else if (thisCon.connType == Connector.
          IS_STUBBORN_CONNECTOR) {
        messageID = new String(message);

        newLine += " " + messageID + " " + commonEnd;
        ArrayList msgDataValue = genMsgData(sent, mepIndex,
            thisCon.iD,
            null, Connector.IS_STUBBORN_CONNECTOR);
        msgData.put(message, msgDataValue);
      } else {
        messageID = new String(message);
```

```
        newLine += " " + messageID + " " + commonEnd;
        ArrayList msgDataValue = genMsgData(sent, mepIndex, thisCon.iD,
            null, Connector.IS_COOPERATIVE_CONNECTOR);
        msgData.put(message, msgDataValue);
      }
      // add to trigger constructor if needed

      if (triggerValue == CSPThreadCounterConstructor.INC_TRIGGER) {
        threadCon.addIncEvent(thisPort.childOf.iD, thisPort.iD,
            messageID);
      }

      if (triggerValue == CSPThreadCounterConstructor.DEC_TRIGGER) {
        threadCon.addDecEvent(thisPort.childOf.iD, thisPort.iD,
            messageID);
      }

      // add message to memory constructor if the port if required
      if (thisPort.isUnicast) {

        if (thisPort.choiceGroupMaker && mepIndex == 1) {
          memCon.addChoiceMaker(thisPort.childOf.iD,
              thisPort.choiceGroup, messageID, otherCompID);
        } else {
          memCon.addChoiceFollower(thisPort.childOf.iD,
              thisPort.choiceGroup, messageID, otherCompID);
        }
      }

    }

    return newLine;
  }

  private static ArrayList genMsgData(boolean sent, int mepIndex,
      String connectorID, String targetID, int connType) {
    ArrayList toReturn = new ArrayList(5);

    // workaround as arraylist size initilzing isnt working
    toReturn.add(null);
    toReturn.add(null);
    toReturn.add(null);
    toReturn.add(null);
    toReturn.add(null);

    if (sent) {
      toReturn.set(0, "sent");
    } else {
      toReturn.set(0, "recv");
    }

    toReturn.set(1, new Integer(mepIndex));
    toReturn.set(2, connectorID);
    toReturn.set(3, targetID);
    toReturn.set(4, new Integer(connType));

    return toReturn;
  }

  private static String processCentralCSP(String[] compCSPSplit,
      String memoryProcess, String compID, CSPMemoryConstructor memCon,
      boolean componentHasUniCastPort) {
    // get and rename process names from line 0
    String[] line0 = compCSPSplit[0].split(" ");
```

```
    Set uniqueNames = new TreeSet();
    String thisName;
    String theNewProcesses = "";
    // if the comp has unicast ports then we need a memory
    // process
    // process IDs are at 2,4,6,8... etc
    // e.g comp = p1 ||| p2 ||| p3
    if (componentHasUniCastPort) {
      for (int index = 2; index < line0.length; index += 2) {
        thisName = line0[index];
        String newName = thisName.trim() + "_withMemory";
        if (!uniqueNames.contains(newName)) {
          uniqueNames.add(newName);
          theNewProcesses += newName
              + " = "
              + memCon.synchProcessAndMemoryProcess(thisName,
                  compID) + " \n \n";
        }
        line0[index] = newName;
      }

      theNewProcesses += memCon.getComponentMemProcess(compID)
          + " \n";
    }

    // recombine the split original process, adding spaces back
         into line 0
    // and then add the new processes and return to the calling
         process
    // to be included in the data structure.

    String newCentralCSP = "";

    newCentralCSP += line0[0];

    for (int index = 1; index < line0.length; index++) {
      newCentralCSP += " " + line0[index];
    }

    newCentralCSP += " \n";

    for (int index = 1; index < compCSPSplit.length; index++) {
      newCentralCSP += compCSPSplit[index] + " \n";
    }

    newCentralCSP += theNewProcesses;
    return newCentralCSP;
  }
}
```

F.4.15 CSP Thread Counter Constructor

```
package uk.ac.ncl.cjg.ws_enhanced.common;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;

public class CSPThreadCounterConstructor {

  public static final int NOT_TRIGGER = 0;
  public static final int INC_TRIGGER = 1;
  public static final int DEC_TRIGGER = 2;
  public static final String ThreadCounterProcess = "
      ThreadCounterProcess";
  public static final String ThreadCounterProcessName = "
      ThreadCounterProcessSpec";

  private Map incEvents = new TreeMap();
  private Map decEvents = new TreeMap();

```

```java
// pr1ivate Set incEvents = new TreeSet();
// private Set decEvents = new TreeSet();

public void addIncEvent(String compID, String portID, String
    eventID) {

  Set incSet = getRequiredSet(incEvents, compID, portID);
  incSet.add(eventID);
}

public void addDecEvent(String compID, String portID, String
    eventID) {
  Set decSet = getRequiredSet(decEvents, compID, portID);
  decSet.add(eventID);
}

public String getCounterTriggersForPort(String compID, String
     portID) {
  Set incSet = getRequiredSet(incEvents, compID, portID);
  Set decSet = getRequiredSet(decEvents, compID, portID);

  String triggers = "";

  Iterator it = incSet.iterator();
  triggers += (String)it.next();
  while(it.hasNext())
  {
    triggers += ", " + (String)it.next();
  }

  it = decSet.iterator();
  while(it.hasNext())
  {
    triggers += ", " + (String)it.next();
  }

  return triggers;
}

public String getCounterProcess(String compID, String portID)
     {

  Set incSet = getRequiredSet(incEvents, compID, portID);
  Set decSet = getRequiredSet(decEvents, compID, portID);

  String theProcess = getCounterProcessName() + " = ";

  Iterator incIt = incSet.iterator();
  theProcess += (String) incIt.next() + " -> " +
      getCounterProcessName()
      + "1 \n";

  while (incIt.hasNext()) {
    theProcess += "    [] " + (String) incIt.next() + " -> "
        + getCounterProcessName() + "1 \n";
  }

  Iterator decIt = decSet.iterator();
  while (decIt.hasNext()) {
    theProcess += "    [] " + (String) decIt.next() + " -> "
        + getCounterProcessName() + " \n";
  }

  theProcess += getCounterProcessName() + "1 = ";

  incIt = incSet.iterator();
  theProcess += (String) incIt.next() + " -> " +
      getCounterProcessName()
      + "2 \n";

  while (incIt.hasNext()) {
    theProcess += "    [] " + (String) incIt.next() + " -> "
```

```java
            + getCounterProcessName() + "2 \n";
    }

    decIt = decSet.iterator();
    while (decIt.hasNext()) {
      theProcess += "    [] " + (String) decIt.next() + " -> "
            + getCounterProcessName() + " \n";
    }

    theProcess += getCounterProcessName() + "2 = Max -> STOP ";

    return theProcess;
  }

  public String getCounterProcessName() {
    return ThreadCounterProcess;
  }

  public String getCounterSpecName() {
    return ThreadCounterProcessName;
  }

  public String getCounterSpec(String compID, String portID,
      CSPHidingSetConstructor hidCon) {

    Set allEvents = hidCon.getAllMessagesAndTriggers();

    String theSpec = getCounterSpecName() + " = ";

    Iterator eventIt = allEvents.iterator();
    theSpec += (String) eventIt.next() + " -> " +
        getCounterSpecName()
        + " \n";

    while (eventIt.hasNext()) {
      theSpec += "    [] " + (String) eventIt.next() + " -> "
            + getCounterSpecName() + " \n";
    }

    return theSpec;
  }

  private Set getRequiredSet(Map parentMap, String compID,
      String portID) {
    if (!parentMap.containsKey(compID)) {
      parentMap.put(compID, new TreeMap());
    }
    Map theComp = (Map) parentMap.get(compID);

    if (!theComp.containsKey(portID)) {
      theComp.put(portID, new TreeSet());
    }
    return (Set) theComp.get(portID);
  }
}
```

F.4.16 Data Extraction Utils

```java
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import org.acmestudio.acme.core.type.IAcmeEnumValue;
import org.acmestudio.acme.core.type.IAcmeRecordField;
import org.acmestudio.acme.core.type.IAcmeRecordValue;
import org.acmestudio.acme.core.type.IAcmeSetValue;
import org.acmestudio.acme.core.type.IAcmeStringValue;
import org.acmestudio.acme.element.IAcmeComponent;
```

```java
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.element.property.IAcmeProperty;

/**
 * This class contains utilities to handle extracting the data from the ACME
 * model. This is to encourage reuse and make the analysis more clear
 */
public class DataExtractionUtils {

  public static String getPortCSP(IAcmePort thePort) throws Exception {

    IAcmeProperty portCSP = thePort.getProperty("MessagePattern");
    if (portCSP == null)
      throw new ReportableException("The port has no CSP property");

    String tempDebug = "****dataextract : getPortCsp**** \n";
    tempDebug += ((IAcmeStringValue) (portCSP.getValue())).getValue() + "\n \n";
    tempDebug += ((IAcmeStringValue) (portCSP.getValue())).getValue().trim() + "\n \n";
    Helper.writeDebug(tempDebug);
    return ((IAcmeStringValue) (portCSP.getValue())).getValue().trim();

  }

  public static List getMessageNamesFromCSP(IAcmeStringValue theCSP)
      throws Exception {
    // this version accepts the raw AcmeStringValue
    // and calls the version that takes the CSP String
    // itself
    return getMessageNamesFromCSP((theCSP.getValue()));

  }

  public static String getMessageNameFromCSPAtLine(String theCSP,
      int lineNumber) throws Exception {

    String[] cspLines = theCSP.split("\n");
    Helper.writeDebug(cspLines + "\n \n");
    if (cspLines.length > lineNumber)
      return getNameFromCSPLine(cspLines[lineNumber]);
    else
      return null;
  }

  public static List getMessageNamesFromCSP(String theCSP)
      throws Exception {

    String[] cspLines = theCSP.split("\n");

    List nameList = new ArrayList();

    // extract first line, this tells us the pattern type
    String patternType = cspLines[0];

    if (patternType.equalsIgnoreCase("noti")) {
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_NOTI_SENDREQ]).trim()));
      Helper.writeDebug(nameList.toString());
      return nameList;
    }

    if (patternType.equalsIgnoreCase("ino")) {
```

```
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_INO_GETREQ]).trim()));
      Helper.writeDebug(nameList.toString());
      return nameList;
    }

    if (patternType.equalsIgnoreCase("roo")) {
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_ROO_SENDREQ]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_ROO_GETFAULT]).trim()));
      Helper.writeDebug(nameList.toString());
      return nameList;
    }

    if (patternType.equalsIgnoreCase("rio")) {
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_RIO_GETREQ]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_RIO_SENDFAULT]).trim()));
      Helper.writeDebug(nameList.toString());
      return nameList;
    }

    if (patternType.equalsIgnoreCase("soli")) {
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_SOLI_SENDREQ]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_SOLI_GETRES]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_SOLI_GETFAULT]).trim()));
      Helper.writeDebug(nameList.toString());
      return nameList;
    }

    if (patternType.equalsIgnoreCase("reqr")) {
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_REQR_GETREQ]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_REQR_SENDRES]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_REQR_SENDFAULT]).trim()));
      Helper.writeDebug(nameList.toString());
      return nameList;
    }

    if (patternType.equalsIgnoreCase("ooi")) {
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_OOI_SENDREQ]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_OOI_GETRES]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_OOI_GETFAULT]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_OOI_SENDFAULT2]).trim()));
      Helper.writeDebug(nameList.toString());
      return nameList;
    }

    if (patternType.equalsIgnoreCase("ioo")) {
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_IOO_GETREQ]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_IOO_SENDRES]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_IOO_SENDFAULT]).trim()));
      nameList.add(getNameFromCSPLine((String)(cspLines[LookUP.CSP_INDEX_IOO_GETFAULT2]).trim()));
      Helper.writeDebug(nameList.toString());
      return nameList;
    }
```

```java
    throw new ReportableException("The CSP pattern type was not recognised");
  }

  public static String getPatternTypeFromCSP(String theCSP) throws Exception {

    String[] cspLines = theCSP.split("\n");
    String thePattern = cspLines[0];
    return thePattern.trim();

  }

  private static String getNameFromCSPLine(String theLine) throws Exception {

    // the message names are the second token on any line
    // that has a name
    // the name is always the 3rd token on the message lines of the template
    // this means index 2 of the split line
    // the trim was added to remove any odd white spaces added in ACME as these get counted as tokens
    Helper.writeDebug(theLine + " \n");
    String[] temp = theLine.trim().split(" ");
    Helper.writeDebug(temp[2] + " \n");
    return temp[2];
  }

  public static TSafeBoolean getSendsFirstMessage(IAcmePort thePort)
      throws Exception {
    IAcmeProperty sendsFirstProperty = thePort
        .getProperty("SendsFirstMessage");
    String theValue = ((IAcmeEnumValue) sendsFirstProperty.getValue())
        .getValue();

    if (theValue.equalsIgnoreCase("yes")) {
      return new TSafeBoolean(TSafeBoolean.YES);
    } else if (theValue.equalsIgnoreCase("no")) {
      return new TSafeBoolean(TSafeBoolean.NO);
    }

    return new TSafeBoolean(TSafeBoolean.UNDEFINED);
  }

  public static Set getMessageSet(IAcmePort thePort) throws Exception {
    IAcmeProperty messages = thePort.getProperty("Messages");
    if (messages == null)
      throw new ReportableException("Port has no messages property");
    return ((IAcmeSetValue) messages.getValue()).getValues();
  }

  public static IAcmeRecordValue getMessageFromSet(Set messageSet,
      String msgName) throws Exception {
    Iterator i = messageSet.iterator();
    while (i.hasNext()) {
      IAcmeRecordValue thisMessage = (IAcmeRecordValue) i.next();
      if (getMessageIDFromMessage(thisMessage).equalsIgnoreCase(msgName)) {
        return thisMessage;
      }
    }
    throw new ReportableException("Message id" + msgName
        + " was not found in the message set");
  }
```

```java
public static IAcmeRecordValue getMessageFromPort(String messageID,
    IAcmePort thePort) throws Exception {
  Set theMessageSet = getMessageSet(thePort);
  return getMessageFromSet(theMessageSet, messageID);
}

public static int getNumberOfDatumInMessage(IAcmeRecordValue theMessage)
    throws Exception {
  // message is a record, the final part of which is a set of datum, it
  // is the cardinality of this set (MessageData) that we need to find
  Set messageDataSet = getMessageDataSetFromMessage(theMessage);
  return messageDataSet.size();
}

public static IAcmeRecordValue getTMessageDatumFromMessageAtIndex(
    IAcmeRecordValue theMessage, int index) throws Exception {
  Set messageDataSet = getMessageDataSetFromMessage(theMessage);
  Iterator i = messageDataSet.iterator();
  int counter = 0;
  while (i.hasNext()) {
    IAcmeRecordValue thisDatum = (IAcmeRecordValue) i.next();
    if (counter == index) {
      return thisDatum;
    }
    counter++;
  }
  throw new ReportableException("There is no TMessageDatum at index "
      + index);
}

public static String getDatumIDFromTMessageDatum(
    IAcmeRecordValue theMessageDatum) throws Exception {
  IAcmeRecordField theIDField = theMessageDatum.getField("DatumId");
  if (theIDField == null)
    throw new ReportableException(
        "A datum in a message does not have a DatumID");
  IAcmeStringValue theID = (IAcmeStringValue) (theIDField.getValue());
  return theID.getValue();
}

public static TDataRep getTDataRepFromTMessageDatum(
    IAcmeRecordValue theMessageDatum) throws Exception {
  IAcmeRecordField theField = theMessageDatum.getField("DatumRep");
  if (theField == null)
    throw new ReportableException(
        "A TMessageDatum in a message has no Datum Rep defined");
  IAcmeEnumValue theDataRep = (IAcmeEnumValue) (theField.getValue());
  return new TDataRep(theDataRep);
}

private static Set getMessageDataSetFromMessage(IAcmeRecordValue theMessage)
    throws Exception {
  Helper.debug("debug-ingetMessageDataSetFromMessage", "theMessage"
      + theMessage);
  IAcmeRecordField fieldContainingSet = theMessage
      .getField("MessageData");
```

```
    if (fieldContainingSet == null)
      throw new ReportableException(
          "A message does not have a MessageDataProperty");
    IAcmeSetValue propertyMessageDataSet = (IAcmeSetValue) (
        fieldContainingSet
        .getValue());
    return propertyMessageDataSet.getValues();
  }

  public static String getMessageIDFromMessage(IAcmeRecordValue
      theRecord)
      throws Exception {
    IAcmeRecordField msgIDField = theRecord.getField("MessageId
        ");
    if (msgIDField == null)
      throw new ReportableException("A message has a null
          MessageID");
    return ((IAcmeStringValue) msgIDField.getValue()).getValue
        ();
  }

  public static Set getCentralDataRecordsFromComponent(
      IAcmeComponent theComponent) throws Exception {
    IAcmeProperty theProperty = theComponent
        .getProperty("CentralDataRecords");
    if (theProperty == null)
      throw new ReportableException(
          "The component has no CentralDataRecordsProperty");
    IAcmeSetValue thePropertyValue = (IAcmeSetValue)
        theProperty.getValue();
    return thePropertyValue.getValues();
  }

  public static IAcmeRecordValue
      getCentralDataRecordFromRecords(
      String datumID, Set theRecords) throws Exception {
    Iterator i = theRecords.iterator();

    while (i.hasNext()) {
      IAcmeRecordValue thisRecord = (IAcmeRecordValue) i.next()
          ;
      if (getDataIDFromCentralDataRecord(thisRecord).
          equalsIgnoreCase(
          datumID)) {
        return thisRecord;
      }
    }
    throw new ReportableException("No CentralDataRecord found
        with ID "
        + datumID);
  }

  public static String getDataIDFromCentralDataRecord(
      IAcmeRecordValue theRecord) throws Exception {
    IAcmeRecordField theField = theRecord.getField("DatumID");
    if (theField == null)
      throw new ReportableException("A CentralDataRecord has no
          DatumID");
    return ((IAcmeStringValue) (theField.getValue())).getValue
        ();
  }

  public static TDataSemantics
      getDataSemanticsFromCentralDataRecord(
      IAcmeRecordValue theRecord) throws Exception {
    IAcmeRecordField theField = theRecord.getField("
        DatumSemantics");
    if (theField == null)
      throw new ReportableException(
          "A CentralDataRecord has not DatumSemantics");
    return new TDataSemantics(theField.getValue());
  }
```

```java

public static TDataRep getDataRepFromMessage(IAcmeRecordValue
        theMessage,
      String datumID) throws Exception {

    IAcmeRecordField datumSetRecordField = theMessage
        .getField("MessageData");
    if (datumSetRecordField == null)
      throw new ReportableException(
          "A message has no MessageData Property");
    IAcmeSetValue datumSetValue = (IAcmeSetValue)
        datumSetRecordField
        .getValue();
    Set datumSet = datumSetValue.getValues();

    // iterate through the set to find the require datum

    Iterator i = datumSet.iterator();

    IAcmeRecordValue theDatumRecord = null;

    while (i.hasNext()) {
      IAcmeRecordValue thisDatumRecord = (IAcmeRecordValue) i.
          next();

      String thisDatumName = getDatumIDFromTMessageDatum(
          thisDatumRecord);
      if (thisDatumName.equalsIgnoreCase(datumID)) {
        return getTDataRepFromTMessageDatum(thisDatumRecord);
      }
    }
    throw new ReportableException("There was no datum found
        with ID "
        + datumID + " in a message");
}

  public static TDataSemantics getDatumSemanticsFromComponent(
      String datumID,
      IAcmeComponent theComponent) throws Exception {
    Set centralDataRecords = getCentralDataRecordsFromComponent
        (theComponent);
    IAcmeRecordValue datumRecord =
        getCentralDataRecordFromRecords(datumID,
        centralDataRecords);
    return getDataSemanticsFromCentralDataRecord(datumRecord);
  }
}
```

F.4.17 Element CSP Data

```java
package uk.ac.ncl.cjg.ws_enhanced.common;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

public class ElementCSPData {

  private Map allElements;

  public ElementCSPData() {
    allElements = new TreeMap();
  }

  public void addComponent(String compID, String theCSP) {

    if (!allElements.containsKey(compID)) {
      ArrayList temp = new ArrayList(2);

      //workround as arraylist size initialization not working
      temp.add(null);
```

```java
      temp.add(null);

      temp.set(0, theCSP);
      temp.set(1, new TreeMap());
      allElements.put(compID, temp);
    }
    else
    {
      ArrayList compData = (ArrayList)allElements.get(compID);
      compData.set(0, theCSP);
    }
  }

  public void addPort(String compID, String portID, String portCSP) {
    addComponent(compID, null);

    // check for template id on first line, and strip it
    String[] cspLines = portCSP.split("\n");
    String[] line0 = cspLines[0].split(" ");
    String cspToAdd="";
    if(line0.length == 1)
    {
      for(int i=1; i<cspLines.length; i++)
      {
        cspToAdd += cspLines[i] + "\n";
      }
    }
    else
    {
      for(int i=0; i<cspLines.length; i++)
      {
        cspToAdd += cspLines[i] + "\n";
      }
    }

    ArrayList compData = getCompData(compID);
    if (compData == null) {
      addComponent(compID, null);
      compData = getCompData(compID);
    }

    Map portMap = (Map) compData.get(1);
    portMap.put(portID, cspToAdd);
  }

  private ArrayList getCompData(String compID) {
    if (allElements.containsKey(compID)) {
      return (ArrayList) allElements.get(compID);
    }
    return null;
  }

  public String getAllElements() {

    String toReturn = "";

    Set compIDs = allElements.keySet();
    Iterator compIt = compIDs.iterator();
    while (compIt.hasNext()) {
      // add component
      String thisCompID = (String)compIt.next();
      ArrayList compData = (ArrayList) allElements.get(thisCompID);
      toReturn += (String) compData.get(0) + "\n\n";

      // add ports
      Map portsMap = (Map) compData.get(1);
      Set portIDs = portsMap.keySet();
      Iterator portIt = portIDs.iterator();

      while (portIt.hasNext()) {
```

```java
        String portKey = (String) portIt.next();
        String portCSP = (String) portsMap.get(portKey);
        toReturn += portCSP + " \n \n ";
      }
    }
    return toReturn;
  }

  public Set getCompIDs()
  {
    return allElements.keySet();
  }

}
```

F.4.18 FDR Results Analyzer

```java
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class FDRResultsAnalyzer {

  private List<List> deadLockTraces;
  private List<List> refinementTraces;
  private boolean deadLockFailed;
  private boolean refinementFailed;
  private boolean analysisCheckFailed;
  private boolean analysisCheckProcessed;
  private String analysisCheckDetails;
  private int selectedAnalysis;
  private CSPHidingSetConstructor hidCon;
  private CSPConnectorConstructor connCon;
  private String compID;

  public FDRResultsAnalyzer(int selectedAnalysis,
      CSPHidingSetConstructor hidCon, String compID,
      CSPConnectorConstructor connCon) {
    deadLockTraces = new LinkedList<List>();
    refinementTraces = new LinkedList<List>();
    analysisCheckProcessed = false;
    analysisCheckDetails = "";
    this.hidCon = hidCon;
    this.compID = compID;
    this.selectedAnalysis = selectedAnalysis;
    this.connCon = connCon;
  }

  public void submitRefinementTraces(List<String> fdrResults) {
    int index = 0;
    Iterator<String> it = fdrResults.iterator();

    // get boolean result
    while (it.hasNext() && index < 3) {
      index++;
      it.next();
    }

    String fdrResult = it.next();

    if (fdrResult.trim().equalsIgnoreCase("xfalse")
        || fdrResult.trim().equalsIgnoreCase("false")) {
      refinementFailed = true;
      refinementTraces = readResults(it);
    } else {
```

```
      refinementFailed = false;
    }
  }

  public void submitDeadlockTraces(List<String> fdrResults) {

    int index = 0;

    Iterator<String> it = fdrResults.iterator();

    // get boolean result
    while (it.hasNext() && index < 3) {

      index++;
      it.next();
    }

    String fdrResult = it.next();
    if (fdrResult.trim().equalsIgnoreCase("xfalse")
        || fdrResult.trim().equalsIgnoreCase("false")) {
      deadLockFailed = true;
      deadLockTraces = readResults(it);
    } else {
      deadLockFailed = false;
    }
  }

  private List<List> readResults(Iterator theResults) {
    List thisTrace = null;
    List<List> examples = new LinkedList();
    while (theResults.hasNext()) {
      String thisLine = (String) theResults.next();
      // assumes the first line with be a BEGIN TRACE
      boolean endTrace = false;
      if (thisLine.startsWith("BEGIN TRACE")) {
        thisTrace = new LinkedList<String>();
        examples.add(thisTrace);
      } else {
        if (!thisLine.startsWith("END TRACE")) {
          thisTrace.add(thisLine.trim());
        } else {
          // do nothing for an end trace line
        }
      }
    }
    return examples;
  }

  public Boolean reportResult() throws ReportableException {

    if (analysisCheckProcessed) {

      return analysisCheckFailed;
    } else {

      if (selectedAnalysis == CSPModelBuilder.
          ANALYSIS_DEADLOCK_PARTIAL) {
        processDeadLockCheck(true);
      }

      if (selectedAnalysis == CSPModelBuilder.ANALYSIS_DEADLOCK
          ) {
        processDeadLockCheck(false);
      }

      if (selectedAnalysis == CSPModelBuilder.
          ANALYSIS_THREAD_SPEC_REFINEMENT) {
        processThreadCheck();
      }

      if (selectedAnalysis == CSPModelBuilder.
          ANALYSIS_COMPONENT_REFINEMENT) {
```

```
      processOmissionCheck(false);
    }

    if (selectedAnalysis == CSPModelBuilder.
        ANALYSIS_COMPONENT_REFINEMENT_PARTIAL) {
      processOmissionCheck(true);
    }
    return analysisCheckFailed;
  }
}

public String reportDetails() throws ReportableException {

  if (analysisCheckProcessed) {

    return analysisCheckDetails;
  } else {

    if (selectedAnalysis == CSPModelBuilder.
        ANALYSIS_DEADLOCK_PARTIAL) {
      processDeadLockCheck(true);
    }

    if (selectedAnalysis == CSPModelBuilder.ANALYSIS_DEADLOCK
        ) {
      processDeadLockCheck(false);
    }

    if (selectedAnalysis == CSPModelBuilder.
        ANALYSIS_THREAD_SPEC_REFINEMENT) {
      processThreadCheck();
    }

    if (selectedAnalysis == CSPModelBuilder.
        ANALYSIS_COMPONENT_REFINEMENT) {
      processOmissionCheck(false);
    }

    if (selectedAnalysis == CSPModelBuilder.
        ANALYSIS_COMPONENT_REFINEMENT_PARTIAL) {
      processOmissionCheck(true);
    }
    return analysisCheckDetails;
  }
}

private void processDeadLockCheck(boolean forPartialMatch)
    throws ReportableException {

  String details = "";
  int failureCount = 0;

  // go through each trace searching for those that end in
  // a message in this components interface
  Set thisCompsMsgs = hidCon.getSetMessagesForComp(compID);

  Iterator traceIt = deadLockTraces.iterator();

  while (traceIt.hasNext()) {
    String temp = "";
    List thisTrace = (List) traceIt.next();
    Iterator thisTraceIt = thisTrace.iterator();
    String thisMessage = "";
    while (thisTraceIt.hasNext()) {
      thisMessage = ((String) thisTraceIt.next()).trim();
      temp += thisMessage + "  ";
    }
    // check if final message is part of this components
        interface

    if (thisCompsMsgs.contains(thisMessage)) {

```

```
        if ((forPartialMatch && connCon
            .isMessageUnderOurControl(thisMessage))
            || (!forPartialMatch && !connCon
                .isMessageUnderOurControl(thisMessage))) {
          analysisCheckFailed = true;
          failureCount++;
          details += " ================================== \n ";
          details += " Commission trace number " + failureCount
              + " \n \n";
          details += temp + " \n \n";
        }
      }
    }

    if (analysisCheckFailed) {
      analysisCheckDetails += compID
          + " attempted to send unexpected messages (commision events) in "
          + failureCount + " traces.";
      analysisCheckDetails += details;
    }
    analysisCheckProcessed = true;
  }

  private void processThreadCheck() {
    // result is based entirely on refinement result
    analysisCheckFailed = refinementFailed;
    if(analysisCheckFailed)
      analysisCheckDetails += "This port experienced two or more simultaneous invocations";
    analysisCheckProcessed = true;
  }

  private void processOmissionCheck(boolean forPartialMatch)
      throws ReportableException {

    Set reducedDeadLocks = reduceDeadLockTraces(deadLockTraces);
    Set reducedRefinements = reduceRefinementTraces(refinementTraces);

    analysisCheckFailed = false;
    String details = "";

    Iterator refineIt = reducedRefinements.iterator();
    while (refineIt.hasNext()) {
      List thisRefinement = (List) refineIt.next();
      // start confident and look for counterexample
      boolean exampleConfident = true;

      // get the last message in the refinement failure and determine
      // if it is under our control, this defines whether it is considered
      // further or not, true is considered for partial match, falst
      // is considered for mismatch.

      String msgID = (String) thisRefinement
          .get(thisRefinement.size() - 1);

      Helper
          .writeDebug("The last message in the refinement was "
              + msgID);
      Helper.writeDebug("This is the list representing the refinement "
          + thisRefinement);

      Helper
          .writeDebug("selected analysis is partial "
```

```java
            + (selectedAnalysis == CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT_PARTIAL));
    Helper
        .writeDebug("selected analysis is mismatch "
            + (selectedAnalysis == CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT));

    // boolean tempBool =
    // ((connCon.isReceivedMessageUnderOurControl(msgID) &&
    // selectedAnalysis ==
    // CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT_PARTIAL)
    // || (!connCon.isReceivedMessageUnderOurControl(msgID) &&
    // selectedAnalysis ==
    // CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT));

    // Helper.writeDebug("results in " +tempBool);

    if ((connCon.isReceivedMessageUnderOurControl(msgID) && selectedAnalysis == CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT_PARTIAL)
        || (!connCon.isReceivedMessageUnderOurControl(msgID) && selectedAnalysis == CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT)) {

      Helper.writeDebug(" Processing this received message ");

      Iterator deadLockIt = reducedDeadLocks.iterator();
      while (deadLockIt.hasNext()) {
        List thisDeadLock = (List) deadLockIt.next();

        Helper
            .writeDebug("  about to check the refinement against this deadlock "
              + thisDeadLock);

        if (deadLockTraceMatchesRefinementHead(thisRefinement,
            thisDeadLock)) {
          Helper
              .writeDebug("     it was found to match so the counter example is found");
          exampleConfident = false;
          break;
        }
      }

      if (exampleConfident) {
        analysisCheckFailed = true;
        details += " =============================== \n";
        details += thisRefinement.toString();
        details += " \n \n";
      }
    }
  }

  analysisCheckDetails += details;
  analysisCheckProcessed = true;
}

private boolean deadLockTraceMatchesRefinementHead(List refinementTrace,
    List deadLockTrace) {

  String dlTrace = "";
  String rfTrace = "";
  Iterator dl = deadLockTrace.iterator();
  while (dl.hasNext()) {
    dlTrace += "," + (String) dl.next();
  }

```

```java
    Iterator rf = refinementTrace.iterator();

    // and odd loop to make sure we add all but the
    // last event to the trace
    String temp = (String) rf.next();
    while (rf.hasNext()) {
      rfTrace += "," + temp;
      temp = (String) rf.next();
    }

    Helper.writeDebug("The rf trace as a string " + rfTrace);
    Helper.writeDebug("The dl trace as a string " + dlTrace);

    if (rfTrace.equals(dlTrace))
      Helper.writeDebug("These traces are found to be equal");
    else
      Helper.writeDebug("These traces are found to be not equal
          ");

    if (rfTrace.equals(dlTrace)) {
      return true;
    } else {
      return false;
    }
  }

  private Set reduceDeadLockTraces(List original) {
    Set reducedDeadLocks = new TreeSet(new TraceComparator());
    Iterator it = original.iterator();
    List thisDeadLock;
    Iterator deadLockIt;
    while (it.hasNext()) {
      boolean newListPopulated = false;
      List temp = new LinkedList();

      thisDeadLock = (List) it.next();
      deadLockIt = thisDeadLock.iterator();
      String thisEvent;
      while (deadLockIt.hasNext()) {
        thisEvent = (String) deadLockIt.next();
        if (!thisEvent.trim().equalsIgnoreCase("_tau")) {
          temp.add(thisEvent);
          newListPopulated = true;
        }
      }

      if (newListPopulated) {
        reducedDeadLocks.add(temp);
      }
    }
    return reducedDeadLocks;
  }

  private Set reduceRefinementTraces(List<List> original) {
    Set reducedRefinements = new TreeSet(new TraceComparator())
        ;

    Iterator it = original.iterator();
    List thisRefinement;
    Iterator refinementIt;
    while (it.hasNext()) {
      boolean newListPopulated = false;
      boolean newListEndsReceivedMessage = false;
      List temp = new LinkedList();

      thisRefinement = (List) it.next();
      refinementIt = thisRefinement.iterator();
      String thisEvent;
      while (refinementIt.hasNext()) {
        thisEvent = (String) refinementIt.next();
        if (!thisEvent.trim().equalsIgnoreCase("_tau")) {
          temp.add(thisEvent);
```

```
        newListPopulated = true;
        if (isInReceivedMessages(thisEvent)) {
          newListEndsReceivedMessage = true;
        } else {
          newListEndsReceivedMessage = false;
        }
      }

    }

    if (newListEndsReceivedMessage && newListPopulated) {
      reducedRefinements.add(temp);
    }
  }
  return reducedRefinements;
}

public class TraceComparator implements Comparator<List> {
  public int compare(List l1, List l2) {
    return l1.toString().compareTo(l2.toString());
  }
}

private boolean isInReceivedMessages(String theEvent) {
  Set msgsData = hidCon.getSetMessagesDataForComp(compID);

  Iterator msgIt = msgsData.iterator();
  ArrayList thisData;
  String thisMsg;
  while (msgIt.hasNext()) {
    thisData = (ArrayList) msgIt.next();
    thisMsg = ((String) thisData.get(0)).trim();
    if (theEvent.trim().equalsIgnoreCase(thisMsg)) {
      return true;
    }
  }
    return false;
  }
}
```

F.4.19 Helper

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.List;

public class Helper {

  private static BufferedWriter currentStream = null;

  public static void debug(String name, String toOutput) {
    try {
      PrintWriter p = new PrintWriter(name);
      p.println(toOutput);
      p.flush();
    } catch (Exception e) {
    }

  }

  public static void openDebug(String fileID) {

    if (currentStream != null) {
      try {
        currentStream.close();
```

```java
      } catch (Exception e) {
        System.err.println("Error write: " + e.getMessage());
      }
    }

    try {
      FileWriter fstream = new FileWriter("/home/carl/" + fileID);
      currentStream = new BufferedWriter(fstream);
    } catch (Exception e) {
      System.err.println("Error write: " + e.getMessage());
    }
  }

  public static void writeDebug(String toWrite) {
    try {
      currentStream.write(toWrite + "\n");
      currentStream.flush();
    } catch (Exception e) {
      System.err.println("Error write: " + e.getMessage());
    }
  }

  public static void closeDebug() {

    try {
      currentStream.close();
    } catch (Exception e) {
      System.err.println("Error write: " + e.getMessage());
    }

  }

  public static List processCSPModel(String outputPath, int maxExamples) {

    List fdrRawResults = new LinkedList();
    String fdrcmd = "fdrBatchMode";
    String fdrExampleMax = "" + maxExamples;
    String modelLocation = outputPath;
    String toRun = fdrcmd + " " + fdrExampleMax + " " + modelLocation;

    try {
      String line;
      Process p = Runtime.getRuntime().exec(toRun);
      BufferedReader input = new BufferedReader(new InputStreamReader(p
          .getInputStream()));
      while ((line = input.readLine()) != null) {
        fdrRawResults.add(line);
      }
      input.close();
    } catch (Exception e) {
      System.err.println("Error execute: " + e.getMessage());
    }
    return fdrRawResults;
  }

  public static void writeModelToFile(String theCSPModel, String outputPath) {
    try {
      FileWriter fstream = new FileWriter(outputPath);
      BufferedWriter out = new BufferedWriter(fstream);
      out.write(theCSPModel);
      out.close();
    } catch (Exception e) {
      System.err.println("Error write: " + e.getMessage());
    }
  }

}
```

F.4.20 Look Up

```
package uk.ac.ncl.cjg.ws_enhanced.common;

public class LookUP {

  // index numbers applied to the messsages the exchanges we
      consider
  public static final int MESSAGE_INDEX_REQUEST = 1;
  public static final int MESSAGE_INDEX_RESPONSE = 2;
  public static final int MESSAGE_INDEX_FAULT = 3;
  public static final int MESSAGE_INDEX_FAULT2 = 4;



  // line numbers where the messages can be found in the
      message exchange
  // pattern templates
  public static final int CSP_INDEX_NOTI_SENDREQ = 1;

  public static final int CSP_INDEX_INO_GETREQ = 1;

  public static final int CSP_INDEX_ROO_SENDREQ = 1;
  public static final int CSP_INDEX_ROO_GETFAULT = 5;

  public static final int CSP_INDEX_RIO_GETREQ = 1;
  public static final int CSP_INDEX_RIO_SENDFAULT = 5;

  public static final int CSP_INDEX_SOLI_SENDREQ = 1;
  public static final int CSP_INDEX_SOLI_GETRES = 3;
  public static final int CSP_INDEX_SOLI_GETFAULT = 4;

  public static final int CSP_INDEX_REQR_GETREQ = 1;
  public static final int CSP_INDEX_REQR_SENDRES = 3;
  public static final int CSP_INDEX_REQR_SENDFAULT = 4;

  public static final int CSP_INDEX_OOI_SENDREQ = 1;
  public static final int CSP_INDEX_OOI_GETRES = 5;
  public static final int CSP_INDEX_OOI_GETFAULT = 4;
  public static final int CSP_INDEX_OOI_SENDFAULT2 = 9;

  public static final int CSP_INDEX_IOO_GETREQ = 1;
  public static final int CSP_INDEX_IOO_SENDRES = 6;
  public static final int CSP_INDEX_IOO_SENDFAULT = 5;
  public static final int CSP_INDEX_IOO_GETFAULT2 = 8;

}
```

F.4.21 Message Comparison

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.acmestudio.acme.core.type.IAcmeRecordValue;
import org.acmestudio.acme.core.type.IAcmeStringValue;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.element.IAcmePort;

public class MessageComparison {

  private static final int UNDER_DATA_1 = 0;
  private static final int UNDER_DATA_2 = 1;
  private static final int OVER_DATA = 2;
  private static final int DATA_TYPES_MATCH = 3;

```

```
public static AnalysisResult messageUnderData1(IAcmePort port1,
    IAcmePort port2, int messageIndex) throws Exception {
  return messageDataAnalysis(port1, port2, messageIndex,
      MessageComparison.UNDER_DATA_1);
}

public static AnalysisResult messageUnderData2(IAcmePort port1,
    IAcmePort port2, int messageIndex) throws Exception {
  return messageDataAnalysis(port1, port2, messageIndex,
      MessageComparison.UNDER_DATA_2);
}

public static AnalysisResult messageOverData(IAcmePort port1,
    IAcmePort port2, int messageIndex) throws Exception {
  return messageDataAnalysis(port1, port2, messageIndex,
      MessageComparison.OVER_DATA);
}

public static AnalysisResult dataTypesMatch(IAcmePort port1,
    IAcmePort port2, int messageIndex) throws Exception {
  return messageDataAnalysis(port1, port2, messageIndex,
      MessageComparison.DATA_TYPES_MATCH);
}

public static AnalysisResult stateScopesMatch(IAcmeConnector conn,
    IAcmePort port1, IAcmePort port2) throws Exception {
  return stateScopeAnalysis(conn, port1, port2);
}

private static AnalysisResult stateScopeAnalysis(IAcmeConnector conn,
    IAcmePort port1, IAcmePort port2) throws Exception {
  // generate an AcmeInterface from one of the ports
  AcmeInterface ai = new AcmeInterface(conn);

  // get parentComponent for each port
  IAcmeComponent comp1 = (IAcmeComponent) port1.getParent();
  IAcmeComponent comp2 = (IAcmeComponent) port2.getParent();
  String comp1ID = comp1.getName();
  String comp2ID = comp2.getName();

  boolean allScopesCompatible = true;
  String reportDetails = "";

  for (int messageIndex = 0; messageIndex < 4; messageIndex++) {
    MessageMapping thisMessageMapping = new MessageMapping(port1,
        port2, messageIndex);

    if (thisMessageMapping.getSentMessage() == null) {
      // no more mappings for this pair of ports, exit the loop
      break;
    }

    // get the message names
    String sentMsgName = ((IAcmeStringValue) (thisMessageMapping
        .getSentMessage()).getField("MessageId").getValue())
        .getValue();
    String recvMsgName = ((IAcmeStringValue) (thisMessageMapping
        .getReceivedMessage()).getField("MessageId").getValue())
        .getValue();

    // get the port names
```

```java
    String sendingPortID = thisMessageMapping.getSendingPort
        ()
        .getName();
    String receivingPortID = thisMessageMapping.
        getReceivingPort()
        .getName();

    // get the component names
    String sendingComponentID = thisMessageMapping
        .getSendingComponent().getName();
    String receivingComponentID = thisMessageMapping
        .getReceivingComponent().getName();

    // get the same elements from the acme interface
    Component sendingComp = null;
    Component receivingComp = null;
    Port sendingPort = null;
    Port receivingPort = null;
    Iterator allElements = ai.elements.iterator();

    while (allElements.hasNext()) {
      Component thisOne = (Component) allElements.next();
      if (thisOne.iD.equalsIgnoreCase(sendingComponentID)) {
        sendingComp = thisOne;
        Iterator portsIt = sendingComp.ports.iterator();
        while (portsIt.hasNext()) {
          Port thisPort = (Port) portsIt.next();
          if (thisPort.iD.equalsIgnoreCase(sendingPortID)) {
            sendingPort = thisPort;
            break;
          }
        }
      }
      if (thisOne.iD.equalsIgnoreCase(receivingComponentID))
          {
        receivingComp = thisOne;
        Iterator portsIt = sendingComp.ports.iterator();
        while (portsIt.hasNext()) {
          Port thisPort = (Port) portsIt.next();
          if (thisPort.iD.equalsIgnoreCase(receivingPortID))
              {
            receivingPort = thisPort;
            break;
          }
        }
      }
    }

    if (sendingComp == null || receivingComp == null
        || sendingPort == null || receivingPort == null) {
      throw new ReportableException(
          "We could not extract the elements required to
              perform this analysis from the acme interface
              model");
    }

    List messageDataMappings = generateMessageDataMappings(
        thisMessageMapping.getSentMessage(),
            thisMessageMapping
            .getSendingComponent(), thisMessageMapping
            .getReceivedMessage(), thisMessageMapping
            .getReceivingComponent());

    // message data mapping allows us to get mappings between
         the sent
    // and
    // received message datum, this forms the link between
        the two
    // components, we can then compare the state scope
        assumptions of
    // each
```

```
    // datum in the message description with the datum description in
    // the
    // opposing component

    // get the required sent message

    Map sentMessage = (Map) sendingPort.messages.get("sentMsgName");
    if (sentMessage == null)
      throw new Exception("Unable to find the message that was sent");

    Iterator messageMappingIt = messageDataMappings.iterator();
    while (messageMappingIt.hasNext()) {
      MessageDataMapping thisMapping = (MessageDataMapping) messageMappingIt
          .next();

      Map sentMessageDatum = (Map) sentMessage.get(thisMapping
          .getSentMsgDatumID());
      String expectedMessageDatumState = (String) sentMessageDatum
          .get("DatumStateScopeExpected");

      Map receivedMessageDatum = (Map) receivingComp.centralDataRecords
          .get(thisMapping.getReceivedMsgDatumID());
      String exhibitedMessageDatumState = (String) receivedMessageDatum
          .get("DatumScopeExhibited");

      if (!StateScopeComparison.exhibitedCompatibleWithExpected(
          exhibitedMessageDatumState, expectedMessageDatumState)) {
        allScopesCompatible = false;
        reportDetails += "The datum "
            + thisMapping.getSentMsgDatumID()
            + " sent in message "
            + sentMsgName
            + " has expected data scope "
            + expectedMessageDatumState
            + ", this is not compatible with the exhibited state "
            + exhibitedMessageDatumState
            + " of the message datum "
            + thisMapping.getReceivedMsgDatumID()
            + " it maps to \n";
      }
    }
  }
  return new AnalysisResult(allScopesCompatible, reportDetails);
}

private static AnalysisResult messageDataAnalysis(IAcmePort port1,
    IAcmePort port2, int messageIndex, int analysisType)
    throws Exception {
  String toReport = "";
  boolean analysisPassedOK = true;

  // get parentComponent for each port
  IAcmeComponent comp1 = (IAcmeComponent) port1.getParent();
  IAcmeComponent comp2 = (IAcmeComponent) port2.getParent();

  // get csp for each port
  String port1CSP = DataExtractionUtils.getPortCSP(port1);
  String port2CSP = DataExtractionUtils.getPortCSP(port2);
```

```java

    // get direction for each port
    TSafeBoolean port1SendsFirst = DataExtractionUtils
        .getSendsFirstMessage(port1);
    TSafeBoolean port2SendsFirst = DataExtractionUtils
        .getSendsFirstMessage(port2);

    MessageMapping thisMessageMapping = new MessageMapping(
        port1, port2,
        messageIndex);

    // generate the MessageDataMapping
    List messageDataMappings = generateMessageDataMappings(
        thisMessageMapping.getSentMessage(), thisMessageMapping
            .getSendingComponent(), thisMessageMapping
            .getReceivedMessage(), thisMessageMapping
            .getReceivingComponent());

    // "UnderData" Analysis
    if (analysisType == MessageComparison.UNDER_DATA_1
        || analysisType == MessageComparison.UNDER_DATA_2) {
      // check for -1 mappings in the received messages
      // indicating expected data missing
      Iterator i = messageDataMappings.iterator();
      while (i.hasNext()) {
        MessageDataMapping thisMapping = (MessageDataMapping) i
            .next();
        if (thisMapping.getReceivedMsgDatumMapping() ==
            MessageDataMapping.DatumNotMatched) {
          String matchingDatumID = searchForMatchingSemantics(
              thisMapping.getReceivedMsgDatumID(),
              thisMessageMapping.getSendingComponent(),
              thisMessageMapping.getReceivingComponent());
          if (analysisType == MessageComparison.UNDER_DATA_1
              && !matchingDatumID.equalsIgnoreCase("")) {
            analysisPassedOK = false;
            toReport += "There is no data in the message sent
                to match "
                + thisMapping.getReceivedMsgDatumID()
                + ", but it does appear to be available in the
                    sending component"
                + " in datum ID " + matchingDatumID +" \n";
          } else if (analysisType == MessageComparison.
              UNDER_DATA_2
              && matchingDatumID.equalsIgnoreCase("")) {
            analysisPassedOK = false;
            toReport += "There is no data in the message sent
                to match "
                + thisMapping.getReceivedMsgDatumID()
                + " and it does not appear to be available in
                    the component. \n";
          }
        }
      }
    }

    if (analysisType == MessageComparison.OVER_DATA) {
      // check for -1 mappings in the sent messages
      // indicating data sent that is not expected
      Iterator i = messageDataMappings.iterator();
      while (i.hasNext()) {
        MessageDataMapping thisMapping = (MessageDataMapping) i
            .next();
        if (thisMapping.getSentMsgDatumMapping() ==
            MessageDataMapping.DatumNotMatched) {
          analysisPassedOK = false;
          toReport += "The following data was sent but is not
              expected: "
              + thisMapping.getSentMsgDatumID() + " \n";
        }
      }
    }
```

```java

    if (analysisType == MessageComparison.DATA_TYPES_MATCH) {
      // check for sent data mappings > -1 then compare
      // the data types of both data items
      Iterator i = messageDataMappings.iterator();
      while (i.hasNext()) {
        MessageDataMapping thisMapping = (MessageDataMapping) i
            .next();
        if (thisMapping.getReceivedMsgDatumMapping() > -1
            && thisMapping.getSentMsgDatumMapping() > -1) {
          TDataRep dataTypeSent = DataExtractionUtils
              .getDataRepFromMessage(thisMessageMapping
                  .getSentMessage(), thisMapping
                  .getSentMsgDatumID());
          TDataRep dataTypeExpected = DataExtractionUtils
              .getDataRepFromMessage(thisMessageMapping
                  .getReceivedMessage(), thisMapping
                  .getReceivedMsgDatumID());

          if (!dataTypeSent.compatibleWith(dataTypeExpected)) {
            analysisPassedOK = false;
            toReport += "The data type ( "
                + dataTypeSent
                + " ) of "
                + thisMapping.getSentMsgDatumID()
                + " in the sent message is not compatible with
                    the data type ("
                + dataTypeExpected + " ) of "
                + thisMapping.getReceivedMsgDatumID()
                + " in the received message. \n";
          }
        }
      }
    }
    return new AnalysisResult(analysisPassedOK, toReport);
  }

  private static List generateMessageDataMappings(
      IAcmeRecordValue sentMessage, IAcmeComponent sendingComp,
      IAcmeRecordValue expectedMessage, IAcmeComponent
          receivingComp)
      throws Exception {
    List theMappings = new ArrayList();

    // first check mappings from sender to receiver, adding -1
        to those
    // with no match.

    int numberDatumSent = DataExtractionUtils
        .getNumberOfDatumInMessage(sentMessage);
    int numberDatumExpected = DataExtractionUtils
        .getNumberOfDatumInMessage(expectedMessage);

    boolean[] sentMatched = new boolean[numberDatumSent];
    boolean[] expectedMatched = new boolean[numberDatumExpected
        ];
    for (int i = 0; i < numberDatumSent; i++)
      sentMatched[i] = false;
    for (int i = 0; i < numberDatumExpected; i++)
      expectedMatched[i] = false;

    // loop to compare all datum in the two messages

    for (int sentIdx = 0; sentIdx < numberDatumSent; sentIdx++)
        {
      IAcmeRecordValue sentMessageDatum = DataExtractionUtils
          .getTMessageDatumFromMessageAtIndex(sentMessage,
              sentIdx);
      String sentDatumID = DataExtractionUtils
          .getDatumIDFromTMessageDatum(sentMessageDatum);

```

```java
      for (int expectedIdx = 0; expectedIdx <
          numberDatumExpected; expectedIdx++) {
        if (!sentMatched[sentIdx] && !expectedMatched[
            expectedIdx]) {

          // get IDs of the sent / received Datum

          TDataSemantics sentSemantics = DataExtractionUtils
              .getDatumSemanticsFromComponent(sentDatumID,
                  sendingComp);

          IAcmeRecordValue expectedMessageDatum =
              DataExtractionUtils
              .getTMessageDatumFromMessageAtIndex(
                  expectedMessage, expectedIdx);
          String expectedDatumID = DataExtractionUtils
              .getDatumIDFromTMessageDatum(expectedMessageDatum
                  );
          TDataSemantics expectedSemantics =
              DataExtractionUtils
              .getDatumSemanticsFromComponent(expectedDatumID,
                  receivingComp);

          if (sentSemantics.compatibleWith(expectedSemantics))
              {
            sentMatched[sentIdx] = true;
            expectedMatched[expectedIdx] = true;

            theMappings.add(new MessageDataMapping(sentDatumID,
                expectedIdx, expectedDatumID, sentIdx));
          }
        }
      }
      // check if the sent data was matched, add a
          messagedatamapping to
      // say this

      if (!sentMatched[sentIdx]) {
        theMappings.add(new MessageDataMapping(sentDatumID, -1,
            "noMatch", sentIdx));
      }
    }

    // map any unmapped receiver datum to -1
    for (int expectedIdx = 0; expectedIdx < numberDatumExpected
        ; expectedIdx++) {
      if (!expectedMatched[expectedIdx]) {
        IAcmeRecordValue expectedMessageDatum =
            DataExtractionUtils
            .getTMessageDatumFromMessageAtIndex(expectedMessage
                ,
                expectedIdx);
        String expectedDatumID = DataExtractionUtils
            .getDatumIDFromTMessageDatum(expectedMessageDatum);
        theMappings.add(new MessageDataMapping("noMatch",
            expectedIdx,
            expectedDatumID, -1));
      }
    }
    return theMappings;
  }

  private static String getSentMessageNameForIndex(String
      theCSP, int theIndex)
      throws Exception {
    List theMessages = DataExtractionUtils.
        getMessageNamesFromCSP(theCSP);
    Iterator i = theMessages.iterator();
    int counter = 1;

    while (i.hasNext()) {
      String theMessageName = (String) i.next();
```

```java
      if (counter == theIndex) {
        return theMessageName;
      }
      counter++;
    }
    return null;
  }

  private static String searchForMatchingSemantics(String
      datumID,
      IAcmeComponent componentToSearch, IAcmeComponent
          receivingComponent)
      throws Exception {
    Set receivingCentralData = DataExtractionUtils
        .getCentralDataRecordsFromComponent(receivingComponent)
            ;
    IAcmeRecordValue firstDataRecord = DataExtractionUtils
        .getCentralDataRecordFromRecords(datumID,
            receivingCentralData);
    TDataSemantics semanticsToFind = DataExtractionUtils
        .getDataSemanticsFromCentralDataRecord(firstDataRecord)
            ;

    Set centralDataToSearch = DataExtractionUtils
        .getCentralDataRecordsFromComponent(componentToSearch);

    Iterator i = centralDataToSearch.iterator();

    while (i.hasNext()) {
      IAcmeRecordValue thisRecord = (IAcmeRecordValue) i.next()
          ;

      if (DataExtractionUtils.
          getDataSemanticsFromCentralDataRecord(
          thisRecord).compatibleWith(semanticsToFind)) {
        return DataExtractionUtils
            .getDataIDFromCentralDataRecord(thisRecord);
      }
    }
    return "";
  }
}
```

F.4.22 Message Data Mapping

```java
package uk.ac.ncl.cjg.ws_enhanced.common;

public class MessageDataMapping {

  public static final int DatumNotMatched = -1;

  private String sentMsgDatumID, receivedMsgDatumID;
  private int sentMsgDatumMapsToReceivedMsgIndex,
      receivedMsgDatumMapsToSentMsgIndex;

  public MessageDataMapping(String sentMsgDatumID,
      int sentMsgDatumMapsToReceivedMsgIndex, String
          msg2DatumID,
      int receivedMsgDatumMapsToSentMsgIndex) {
    this.sentMsgDatumID = sentMsgDatumID;
    this.sentMsgDatumMapsToReceivedMsgIndex =
        sentMsgDatumMapsToReceivedMsgIndex;
    this.receivedMsgDatumID = msg2DatumID;
    this.receivedMsgDatumMapsToSentMsgIndex =
        receivedMsgDatumMapsToSentMsgIndex;
  }

  public String getSentMsgDatumID() {
    return sentMsgDatumID;
  }

  public String getReceivedMsgDatumID() {
```

```
    return receivedMsgDatumID;
  }

  public int getSentMsgDatumMapping() {
    return sentMsgDatumMapsToReceivedMsgIndex;
  }

  public int getReceivedMsgDatumMapping() {
    return receivedMsgDatumMapsToSentMsgIndex;
  }

}
```

F.4.23 Message Data Types Match

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.AcceptableException;
import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.MessageComparison;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class MessageDataTypesMatch implements IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis
    Wait.delayAnalysis();


    // extract data types from analysis call, this should be passed two
    // ports and
    // an integer
    String ruleID = null;
    String ruleIDNoMessageNumber = "ActiveAnalysisMessageDataTypesMatch";
    IAcmeConnector theElement = null;
    IAcmePort port1 = null;
    IAcmePort port2 = null;
    Integer theMessageIndex = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeConnector) i.next();
      port1 = (IAcmePort) i.next();
      port2 = (IAcmePort) i.next();
      theMessageIndex = (Integer) i.next();
```

```
      ruleID = ruleIDNoMessageNumber + "-msg" + theMessageIndex;
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try{
    if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(
        ruleIDNoMessageNumber, theElement)) {
      Reporter.report(theElement, ruleID, "");
      return Boolean.TRUE;
    }
    } catch (ReportableException rE){
      Reporter
      .report(
          theElement,
          ruleID,
          "There was a reportable Exception raised when getting the activity status of this analysis: \n",
          rE);
  return Boolean.FALSE;

    } catch (Exception e){
      Reporter
      .report(
          theElement,
          ruleID,
          "There was a general Exception raised when getting the activity status of this analysis: \n",
          e);
  return Boolean.FALSE;
    }

    // perform the analysis
    try {
      theResult = MessageComparison.dataTypesMatch(port1, port2,
          theMessageIndex);
    } catch (AcceptableException e) {
      theResult = new AnalysisResult(true, "");
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }
}
```

F.4.24 Message Exchange Patterns Match

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.List;
```

```java
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.MessagePatternComparison;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class MessageExchangePatternsMatch implements
    IExternalAnalysisExpressionNode {

  static int counter = 0;

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis

    Wait.delayAnalysis();

    // extract data types from analysis call, this should be passed
    // a single component
    String ruleID = "ActiveAnalysisMessageExchangePatternsMatch";
    IAcmeConnector theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeConnector) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when getting the activity status of this analysis: \n",
              rE);
      return Boolean.FALSE;
```

```

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a general Exception raised when
                  getting the activity status of this analysis:
                  \n",
              e);
      return Boolean.FALSE;
    }

    // perform the analysis
    try {

      String focusCompID = theElement.getName();

      int comparisonAssessment = MessagePatternComparison
          .compareMessagePatternsInPorts(theElement);

      switch (comparisonAssessment) {
      case MessagePatternComparison.PATTERNS_MATCH:
        theResult = new AnalysisResult(true, "");
        break;
      case MessagePatternComparison.PATTERNS_PARTIALLY_MATCH:
        theResult = new AnalysisResult(false, "These patterns
            partially match thanks to one or more of them being
             in our control domain");
        break;
      case MessagePatternComparison.PATTERNS_MISMATCH:
        theResult = new AnalysisResult(false, "The patterns
            differ and neither port is in our control domain");
        break;
      default:
        theResult = new AnalysisResult(false, "The patterns
            simply do not match due to message passing
            directions");
        break;
      }
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the
                  analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }

}
```

F.4.25 Message Exchange Patterns Partially Match

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.List;
import java.util.Stack;

```

```java
import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.MessagePatternComparison;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class MessageExchangePatternsPartiallyMatch implements
    IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis

    Wait.delayAnalysis();

    // extract data types from analysis call, this should be passed
    // a single component
    String ruleID = "ActiveAnalysisMessageExchangePatternsMatch";
    IAcmeConnector theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeConnector) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when getting the activity status of this analysis: \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
```

```
            theElement,
            ruleID,
            "There was a general Exception raised when getting the activity status of this analysis: \n",
            e);
      return Boolean.FALSE;
    }

    // perform the analysis
    try {

      String focusCompID = theElement.getName();

      int comparisonAssessment = MessagePatternComparison
          .compareMessagePatternsInPorts(theElement);

      switch (comparisonAssessment) {
      case MessagePatternComparison.PATTERNS_MATCH:
        theResult = new AnalysisResult(true, "");
        break;
      case MessagePatternComparison.PATTERNS_PARTIALLY_MATCH:
        theResult = new AnalysisResult(true, "");
        break;
      case MessagePatternComparison.PATTERNS_MISMATCH:
        theResult = new AnalysisResult(false, "The patterns differ and neither port is in our control domain");
        break;
      default:
        theResult = new AnalysisResult(false, "The patterns simply do not match due to message passing directions");
        break;
      }
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }

}
```

F.4.26 Message Mapping

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;

import org.acmestudio.acme.core.type.IAcmeRecordValue;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.element.IAcmePort;

```

```java
public class MessageMapping {

  private IAcmeComponent sendingComp, receivingComp;
  private IAcmePort sendingPort, receivingPort;
  private IAcmeRecordValue sentMessage, receivedMessage;
  private String sentMessageName, receivedMessageName;
  private int messageIndex;

  public static final int NO_MAPPING = -1;

  public MessageMapping(IAcmePort port1, IAcmePort port2, int messageIndex)
      throws Exception {
    int sendCSPIndex = NO_MAPPING;
    int receiveCSPIndex = NO_MAPPING;

    // get the CSP

    String csp1 = DataExtractionUtils.getPortCSP(port1);
    String csp2 = DataExtractionUtils.getPortCSP(port2);

    // get the pattern types
    String cspPattern1 = DataExtractionUtils.getPatternTypeFromCSP(csp1);
    String cspPattern2 = DataExtractionUtils.getPatternTypeFromCSP(csp2);

    // temporary vars until we determine who sends this actual message

    IAcmeComponent sendsFirstComp = null;
    IAcmePort sendsFirstPort = null;
    String sendsFirstCSP = null;
    String sendsFirstCSPPattern = null;
    IAcmeComponent receivesFirstComp = null;
    IAcmePort receivesFirstPort = null;
    String receivesFirstCSP = null;
    String receivesFirstCSPPattern = null;

    String sendingCSP = null;
    String receivingCSP = null;
    // map ports and component to send and receive

    if (isSendFirstPattern(cspPattern1) == true
        && isSendFirstPattern(cspPattern2) == false) {
      sendsFirstComp = (IAcmeComponent) port1.getParent();
      sendsFirstPort = port1;
      sendsFirstCSP = csp1;
      sendsFirstCSPPattern = cspPattern1;
      receivesFirstComp = (IAcmeComponent) port2.getParent();
      receivesFirstPort = port2;
      receivesFirstCSP = csp2;
      receivesFirstCSPPattern = cspPattern2;
    }

    if (isSendFirstPattern(cspPattern1) == false
        && isSendFirstPattern(cspPattern2) == true) {
      sendsFirstComp = (IAcmeComponent) port2.getParent();
      sendsFirstPort = port2;
      sendsFirstCSP = csp2;
      sendsFirstCSPPattern = cspPattern2;
      receivesFirstComp = (IAcmeComponent) port1.getParent();
      receivesFirstPort = port1;
      receivesFirstCSP = csp1;
      receivesFirstCSPPattern = cspPattern1;
    }

    if (isSendFirstPattern(cspPattern1) == isSendFirstPattern(cspPattern2)) {
      throw new ReportableException(
          "Both ports want to send first or both ports want to receive first.");
```

```
    }

    sentMessageName = null;
    sentMessage = null;
    receivedMessageName = null;
    receivedMessage = null;

    // get the required relevant message indexes
    List indexMappingsForThesePatterns = getMessageVectorsForThesePatterns(
        sendsFirstCSPPattern, receivesFirstCSPPattern, sendsFirstCSP,
        receivesFirstCSP);

    if(messageIndex > indexMappingsForThesePatterns.size())
      throw new AcceptableException("This message pairing has no message at this index number");
    Iterator i = indexMappingsForThesePatterns.iterator();
    int counter = 0;
    while (i.hasNext()) {
      counter++;
      MessageVector thisMessageVector = (MessageVector) i.next();
      if (counter == messageIndex) {

        sentMessageName = thisMessageVector.getSentMessageID();
        receivedMessageName = thisMessageVector.getreceivedMessageID();

        if (thisMessageVector.directionIsFromSendsFirst()) {
          sendingComp = sendsFirstComp;
          sendingPort = sendsFirstPort;
          sendingCSP = sendsFirstCSP;

          receivingComp = receivesFirstComp;
          receivingPort = receivesFirstPort;
          receivingCSP = receivesFirstCSP;

          if (thisMessageVector.getSendsFirstCSPIndex() != MessageVector.NO_MAPPING_INDEX) {
            sendCSPIndex = thisMessageVector
                .getSendsFirstCSPIndex();
          }

          if (thisMessageVector.getReceivesFirstCSPIndex() != MessageVector.NO_MAPPING_INDEX) {
            receiveCSPIndex = thisMessageVector
                .getReceivesFirstCSPIndex();
          }
        } else {
          sendingComp = receivesFirstComp;
          sendingPort = receivesFirstPort;
          sendingCSP = receivesFirstCSP;

          receivingComp = sendsFirstComp;
          receivingPort = sendsFirstPort;
          receivingCSP = sendsFirstCSP;

          if (thisMessageVector.getSendsFirstCSPIndex() != MessageVector.NO_MAPPING_INDEX) {
            sendCSPIndex = thisMessageVector
                .getReceivesFirstCSPIndex();
          }

          if (thisMessageVector.getReceivesFirstCSPIndex() != MessageVector.NO_MAPPING_INDEX) {
            receiveCSPIndex = thisMessageVector
                .getSendsFirstCSPIndex();
          }
        }
        sentMessage = DataExtractionUtils.getMessageFromPort(
            sentMessageName, sendingPort);
```

```java
        receivedMessage = DataExtractionUtils.
            getMessageFromPort(
            receivedMessageName, receivingPort);
      }
    }
  }

  private static boolean isSendFirstPattern(String cspPattern)
      {
    // setup map
    Map sendOrReceive = new TreeMap();
    sendOrReceive.put("noti", "send");
    sendOrReceive.put("roo", "send");
    sendOrReceive.put("soli", "send");
    sendOrReceive.put("ooi", "send");
    sendOrReceive.put("ino", "receive");
    sendOrReceive.put("rio", "receive");
    sendOrReceive.put("reqr", "receive");
    sendOrReceive.put("ioo", "receive");

    String patternDir = (String) sendOrReceive
        .get(cspPattern.toLowerCase());

    if (patternDir.equals("send")) {
      return true;
    } else {
      return false;
    }
  }

  private List getMessageVectorsForThesePatterns(String
      senderCSPPattern,
      String receiverCSPPattern, String senderCSP, String
          receiverCSP)
      throws Exception {

    if (senderCSPPattern.equalsIgnoreCase("noti")) {
      if (receiverCSPPattern.equalsIgnoreCase("ino")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_NOTI_SENDREQ,
            LookUP.CSP_INDEX_INO_GETREQ, true, senderCSP,
            receiverCSP));
        return l;
      } else if (receiverCSPPattern.equalsIgnoreCase("rio")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_NOTI_SENDREQ,
            LookUP.CSP_INDEX_RIO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_RIO_SENDFAULT, false, senderCSP,
            receiverCSP));
        return l;
      } else if (receiverCSPPattern.equalsIgnoreCase("reqr")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_NOTI_SENDREQ,
            LookUP.CSP_INDEX_REQR_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_REQR_SENDRES, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_REQR_SENDFAULT, false, senderCSP,
            receiverCSP));
        return l;

      } else {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_NOTI_SENDREQ,
            LookUP.CSP_INDEX_IOO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_IOO_SENDFAULT, false, senderCSP,
```

```java
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_IOO_SENDRES, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_IOO_GETFAULT2, true, senderCSP,
            receiverCSP));
        return l;

      }
    } else if (senderCSPPattern.equalsIgnoreCase("roo")) {
      if (receiverCSPPattern.equalsIgnoreCase("ino")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_ROO_SENDREQ,
            LookUP.CSP_INDEX_INO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_ROO_GETFAULT,
            MessageVector.NO_MAPPING_INDEX, false, senderCSP,
            receiverCSP));
        return l;
      } else if (receiverCSPPattern.equalsIgnoreCase("rio")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_ROO_SENDREQ,
            LookUP.CSP_INDEX_RIO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_ROO_GETFAULT,
            LookUP.CSP_INDEX_RIO_SENDFAULT, false, senderCSP,
            receiverCSP));
        return l;
      } else if (receiverCSPPattern.equalsIgnoreCase("reqr")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_ROO_SENDREQ,
            LookUP.CSP_INDEX_REQR_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_ROO_GETFAULT,
            LookUP.CSP_INDEX_REQR_SENDFAULT, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_REQR_SENDRES, false, senderCSP,
            receiverCSP));
        return l;
      } else {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_ROO_SENDREQ,
            LookUP.CSP_INDEX_IOO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_ROO_GETFAULT,
            LookUP.CSP_INDEX_IOO_SENDFAULT, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_IOO_SENDRES, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_IOO_GETFAULT2, true, senderCSP,
            receiverCSP));
        return l;
      }
    } else if (senderCSPPattern.equalsIgnoreCase("soli")) {
      if (receiverCSPPattern.equalsIgnoreCase("ino")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_SENDREQ,
            LookUP.CSP_INDEX_INO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_GETRES,
            MessageVector.NO_MAPPING_INDEX, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_GETFAULT,
            MessageVector.NO_MAPPING_INDEX, false, senderCSP,
            receiverCSP));
        return l;
      } else if (receiverCSPPattern.equalsIgnoreCase("rio")) {
        List l = new ArrayList();
```

```java
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_SENDREQ,
            LookUP.CSP_INDEX_RIO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_GETFAULT,
            LookUP.CSP_INDEX_RIO_SENDFAULT, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_GETRES,
            MessageVector.NO_MAPPING_INDEX, false, senderCSP,
            receiverCSP));
        return l;
      } else if (receiverCSPPattern.equalsIgnoreCase("reqr")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_SENDREQ,
            LookUP.CSP_INDEX_REQR_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_GETRES,
            LookUP.CSP_INDEX_REQR_SENDRES, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_GETFAULT,
            LookUP.CSP_INDEX_REQR_SENDFAULT, false, senderCSP,
            receiverCSP));
        return l;
      } else {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_SENDREQ,
            LookUP.CSP_INDEX_IOO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_GETRES,
            LookUP.CSP_INDEX_IOO_SENDRES, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_SOLI_GETFAULT,
            LookUP.CSP_INDEX_IOO_SENDFAULT, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(MessageVector.NO_MAPPING_INDEX,
            LookUP.CSP_INDEX_IOO_GETFAULT2, true, senderCSP,
            receiverCSP));
        return l;
      }
    } else {
      if (receiverCSPPattern.equalsIgnoreCase("ino")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_SENDREQ,
            LookUP.CSP_INDEX_INO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_GETFAULT,
            MessageVector.NO_MAPPING_INDEX, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_GETRES,
            MessageVector.NO_MAPPING_INDEX, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_SENDFAULT2
            ,
            MessageVector.NO_MAPPING_INDEX, true, senderCSP,
            receiverCSP));
        return l;
      } else if (receiverCSPPattern.equalsIgnoreCase("rio")) {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_SENDREQ,
            LookUP.CSP_INDEX_RIO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_GETFAULT,
            LookUP.CSP_INDEX_RIO_SENDFAULT, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_GETRES,
            MessageVector.NO_MAPPING_INDEX, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_SENDFAULT2
            ,
            MessageVector.NO_MAPPING_INDEX, true, senderCSP,
            receiverCSP));
        return l;
      } else if (receiverCSPPattern.equalsIgnoreCase("reqr")) {
```

```
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_SENDREQ,
            LookUP.CSP_INDEX_REQR_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_GETFAULT,
            LookUP.CSP_INDEX_REQR_SENDFAULT, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_GETRES,
            LookUP.CSP_INDEX_REQR_SENDRES, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_SENDFAULT2
            ,
            MessageVector.NO_MAPPING_INDEX, true, senderCSP,
            receiverCSP));
        return l;
      } else {
        List l = new ArrayList();
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_SENDREQ,
            LookUP.CSP_INDEX_IOO_GETREQ, true, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_GETFAULT,
            LookUP.CSP_INDEX_IOO_SENDFAULT, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_GETRES,
            LookUP.CSP_INDEX_IOO_SENDRES, false, senderCSP,
            receiverCSP));
        l.add(new MessageVector(LookUP.CSP_INDEX_OOI_SENDFAULT2
            ,
            LookUP.CSP_INDEX_IOO_GETFAULT2, true, senderCSP,
            receiverCSP));
        return l;
      }
    }
  }

  public IAcmeComponent getSendingComponent() {
    return sendingComp;
  }

  public IAcmeComponent getReceivingComponent() {
    return receivingComp;
  }

  public IAcmePort getSendingPort() {
    return sendingPort;
  }

  public IAcmePort getReceivingPort() {
    return receivingPort;
  }

  public IAcmeRecordValue getSentMessage() {
    return sentMessage;
  }

  public IAcmeRecordValue getReceivedMessage() {
    return receivedMessage;
  }

}
```

F.4.27 Message Over Data

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.environment.error.AcmeError;
```

```java
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.AcceptableException;
import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.MessageComparison;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class MessageOverData implements IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis
    Wait.delayAnalysis();

    String ruleID = null;
    String ruleIDNoMessageNumber = "ActiveAnalysisMessageOverData";
    IAcmeConnector theElement = null;
    IAcmePort port1 = null;
    IAcmePort port2 = null;
    Integer theMessageIndex = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeConnector) i.next();
      port1 = (IAcmePort) i.next();
      port2 = (IAcmePort) i.next();
      theMessageIndex = (Integer) i.next();
      ruleID = ruleIDNoMessageNumber + "-msg" + theMessageIndex;
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try{
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(
          ruleIDNoMessageNumber, theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
      } catch (ReportableException rE){
        Reporter
        .report(
            theElement,
            ruleID,
            "There was a reportable Exception raised when getting the activity status of this analysis: \n",
            rE);
    return Boolean.FALSE;

      } catch (Exception e){
        Reporter
```

```
        .report(
            theElement,
            ruleID,
            "There was a general Exception raised when getting
                the activity status of this analysis: \n",
            e);
    return Boolean.FALSE;
      }

    // perform the analysis
    try {
      theResult = MessageComparison.messageOverData(port1,
          port2,
          theMessageIndex);
    } catch (AcceptableException e) {
      theResult = new AnalysisResult(true, "");
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the
                  analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }

}
```

F.4.28 Message Pattern Comparison

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

import org.acmestudio.acme.element.IAcmeConnector;

public class MessagePatternComparison {
  public static final int PATTERNS_MATCH = 0;
  public static final int PATTERNS_PARTIALLY_MATCH = 1;
  public static final int PATTERNS_MISMATCH = 2;
  public static final int PATTERN_DIRECTIONS_WRONG = 3;

  public static int compareMessagePatternsInPorts(
      IAcmeConnector theConnector)
      throws ReportableException, Exception {

    String connectorID = theConnector.getName();

    // build the acme interface
    AcmeInterface ai = new AcmeInterface(theConnector);

    // get the acme interface version of this connector
    Iterator connIT = ai.conns.iterator();
    Connector thisConn = null;
    boolean found = false;
    while (connIT.hasNext()) {
      thisConn = (Connector) connIT.next();
      if (thisConn.iD.equalsIgnoreCase(connectorID)) {
```

```
        found = true;
        break;
      }
    }
    if (!found)
      throw new Exception(
          "No connector with the correct name was found in the
              Acme Interface");

    // get the two ports attached to this connector
    Port p1;
    Port p2;

    if (thisConn.r1 == null)
      throw new ReportableException(
          "Role 1 on this connector does not have a port
              attached");
    else
      p1 = thisConn.r1;

    if (thisConn.r2 == null)
      throw new ReportableException(
          "Role 2 on this connector does not have a port
              attached");
    else
      p2 = thisConn.r2;

    // extract their mep and control status
    String[] p1MEP = p1.messagePattern.split("\n");
    String[] p2MEP = p2.messagePattern.split("\n");
    boolean p1InOurControl = p1.inOurControlDomain;
    boolean p2InOurControl = p2.inOurControlDomain;

    // get the first lines and lookup the match status
    String p1MEPType = p1MEP[0].trim();
    String p2MEPType = p2MEP[0].trim();

    int basicPatternMatch = patternPairLookup(p1MEPType,
        p2MEPType);

    if (basicPatternMatch == PATTERNS_MATCH)
      return PATTERNS_MATCH;

    if (basicPatternMatch == PATTERNS_PARTIALLY_MATCH) {
      if (p1InOurControl || p2InOurControl)
        return PATTERNS_PARTIALLY_MATCH;
      else
        return PATTERNS_MISMATCH;
    }

    // if reaches this point then the directions must be wrong
    return PATTERN_DIRECTIONS_WRONG;
  }

  private static int patternPairLookup(String pattern1, String
      pattern2) {
    List<String> senderPatterns = new LinkedList<String>();
    senderPatterns.add("noti");
    senderPatterns.add("roo");
    senderPatterns.add("soli");
    senderPatterns.add("ooi");

    List<String> receiverPatterns = new LinkedList<String>();
    receiverPatterns.add("ino");
    receiverPatterns.add("rio");
    receiverPatterns.add("reqr");
    receiverPatterns.add("ioo");

    String senderPattern;
    String receiverPattern;

    if (senderPatterns.contains(pattern1.trim().toLowerCase())
```

```java
        && receiverPatterns.contains(pattern2.trim().
            toLowerCase())) {
      senderPattern = pattern1.trim();
      receiverPattern = pattern2.trim();
    } else if (senderPatterns.contains(pattern2.trim().
        toLowerCase())
        && receiverPatterns.contains(pattern1.trim().
            toLowerCase())) {
      senderPattern = pattern2.trim();
      receiverPattern = pattern1.trim();
    } else {
      // this assumes the pattern names have been input
          correctly, either
      // way something is wrong
      return PATTERN_DIRECTIONS_WRONG;
    }

    if (senderPattern.equalsIgnoreCase("noti")) {
      if (receiverPattern.equalsIgnoreCase("ino"))
        return PATTERNS_MATCH;
      else
        return PATTERNS_PARTIALLY_MATCH;
    }

    if (senderPattern.equalsIgnoreCase("roo")) {
      if (receiverPattern.equalsIgnoreCase("rio"))
        return PATTERNS_MATCH;
      else
        return PATTERNS_PARTIALLY_MATCH;
    }

    if (senderPattern.equalsIgnoreCase("soli")) {
      if (receiverPattern.equalsIgnoreCase("reqr"))
        return PATTERNS_MATCH;
      else
        return PATTERNS_PARTIALLY_MATCH;
    }

    // final case, sender must be ooi to reach here
    if (receiverPattern.equalsIgnoreCase("ioo"))
      return PATTERNS_MATCH;
    else
      return PATTERNS_PARTIALLY_MATCH;
  }
}
```

F.4.29 Message Pattern And Message List Concur

```java
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Stack;
import java.util.TreeSet;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.
    IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.
    AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.AcmeInterface;
import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.Component;
import uk.ac.ncl.cjg.ws_enhanced.common.DataExtractionUtils;
import uk.ac.ncl.cjg.ws_enhanced.common.Helper;
import uk.ac.ncl.cjg.ws_enhanced.common.LookUP;
```

```java
import uk.ac.ncl.cjg.ws_enhanced.common.Port;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class MessagePatternAndMessageListConcur implements
    IExternalAnalysisExpressionNode {

  private static final int MSG_SETS_EQUAL = 1;
  private static final int MEP_MSGS_SUPERSET_OF_MESSAGES = 2;
  private static final int MESSAGES_SUPERSET_OF_MEP_MSGS = 3;
  private static final int BOTH_SETS_CONTAIN_UNCOMMON_MESSAGES = 4;
  private static final int SETS_ARE_DISJOINT = 5;

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {
    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis

    Wait.delayAnalysis();

    // extract data types from analysis call, this should be passed
    // a single component
    String ruleID = "ActiveAnalysisMessagePatternAndMessageListConcur";
    IAcmePort theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmePort) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when getting the activity status of this analysis: \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a general Exception raised when getting the activity status of this analysis: \n",
```

```java
              e);
      return Boolean.FALSE;
    }

    // perform the analysis
    try {

      // construct the acme interface and grab the required port from it
      String focusPortID = theElement.getName();
      String focusPortParentComponentID = theElement.getParent()
          .getName();
      AcmeInterface ai = new AcmeInterface(theElement);

      Port thePort = null;
      boolean requiredPortFound = false;
      Iterator allComponents = ai.elements.iterator();
      while (allComponents.hasNext()) {
        Component thisComponent = (Component) allComponents.next();
        if (thisComponent.iD
            .equalsIgnoreCase(focusPortParentComponentID)) {
          Iterator componentPortsIt = thisComponent.ports.iterator();
          while (componentPortsIt.hasNext()) {
            thePort = (Port) componentPortsIt.next();
            if (thePort.iD.equalsIgnoreCase(focusPortID)) {
              requiredPortFound = true;
              break;
            }
          }
        }
        if (requiredPortFound)
          break;
      }

      if (!requiredPortFound)
        throw new ReportableException(
            "The required port was not found in the model");

      // get message structure names from the port
      Set namesFromStructure = getMessageNamesFromMessages(thePort.messages);
      Set namesFromPattern = getMessageNamesFromPattern(thePort.messagePattern);

      CompareListsResult structureFirstCheck = compareLists(
          namesFromStructure, namesFromPattern, true);
      CompareListsResult patternFirstCheck = compareLists(
          namesFromPattern, namesFromStructure, false);

      boolean noMismatchFound;
      if (structureFirstCheck.aMismatchWasFound()
          || patternFirstCheck.aMismatchWasFound())
        noMismatchFound = false;
      else
        noMismatchFound = true;

      boolean commonMessagesFound;
      if (structureFirstCheck.aCommonMessageWasFound()
          || patternFirstCheck.aCommonMessageWasFound())
        commonMessagesFound = true;
      else
        commonMessagesFound = false;

      String reportDetails = structureFirstCheck.getReport()
          + patternFirstCheck.getReport();

      if (!commonMessagesFound)
        reportDetails += "There were no common message names found in the either property ";
```

```java

      String messageListsAsStrings = null;
      if (!noMismatchFound) {
        messageListsAsStrings = "Messages found in Messages property: \n";
        Iterator nameIt = namesFromStructure.iterator();
        while (nameIt.hasNext()) {
          messageListsAsStrings += (String) nameIt.next() + "\n";
        }
        messageListsAsStrings += "Messages found in MessagePattern property: \n";
        nameIt = namesFromPattern.iterator();
        while (nameIt.hasNext()) {
          messageListsAsStrings += (String) nameIt.next() + "\n";
        }

        reportDetails += "\n" + messageListsAsStrings;
      }

      theResult = new AnalysisResult(noMismatchFound,
          reportDetails);

    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }

  private CompareListsResult compareLists(Set l1, Set l2,
      boolean firstListFromMessagesStructure) {
    CompareListsResult thisResult = new CompareListsResult();
    Iterator l1It = l1.iterator();
    while (l1It.hasNext()) {
      boolean thisMsgMatched = false;
      String l1msg = (String) l1It.next();
      Iterator l2It = l2.iterator();
      while (l2It.hasNext()) {
        String l2msg = (String) l2It.next();
        if (l1msg.equalsIgnoreCase(l2msg)) {
          thisMsgMatched = true;
          thisResult.foundACommonMessage();
          break;
        }
      }

      if (!thisMsgMatched) {
        thisResult.foundAMismatch();
        thisResult.addReportLine(" the message " + l1msg
            + " was found in the");
        if (firstListFromMessagesStructure)
          thisResult
              .addReportLine("Messages property but not in the Message Exchange Pattern \n");
        else
```

```java
          thisResult
              .addReportLine("Message Exchange Pattern property
                    but not in the Messages \n");
      }
    }

    return thisResult;
  }

  private Set<String> getMessageNamesFromMessages(Map messages)
      {
    Set<String> messageNames = messages.keySet();
    return messageNames;
  }

  private Set<String> getMessageNamesFromPattern(String
      messagePattern)
      throws ReportableException, Exception {
    String[] patternSplit = messagePattern.split("\n");
    Set<String> messageNames = new TreeSet();

    String pattern = patternSplit[0].trim();

    if (pattern.equalsIgnoreCase("noti")) {
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_NOTI_SENDREQ));
    }

    if (pattern.equalsIgnoreCase("ino")) {
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_INO_GETREQ));
    }

    if (pattern.equalsIgnoreCase("roo")) {
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_ROO_SENDREQ));
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_ROO_GETFAULT));
    }

    if (pattern.equalsIgnoreCase("rio")) {
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_RIO_GETREQ));
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_RIO_SENDFAULT));
    }

    if (pattern.equalsIgnoreCase("reqr")) {
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_REQR_GETREQ));
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_REQR_SENDRES));
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_REQR_SENDFAULT));
    }

    if (pattern.equalsIgnoreCase("soli")) {
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_SOLI_SENDREQ));
      messageNames.add(DataExtractionUtils.
          getMessageNameFromCSPAtLine(
          messagePattern, LookUP.CSP_INDEX_SOLI_GETRES));
```

```java
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_SOLI_GETFAULT));
    }

    if (pattern.equalsIgnoreCase("ooi")) {
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_OOI_SENDREQ));
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_OOI_GETRES));
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_OOI_GETFAULT));
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_OOI_SENDFAULT2));
    }

    if (pattern.equalsIgnoreCase("ioo")) {
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_IOO_GETREQ));
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_IOO_SENDRES));
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_IOO_SENDFAULT));
        messageNames.add(DataExtractionUtils.
            getMessageNameFromCSPAtLine(
            messagePattern, LookUP.CSP_INDEX_IOO_GETFAULT2));
    }

    return messageNames;
  }

  private class CompareListsResult {
    private boolean aMismatchWasFound;
    private boolean aCommonMessageWasFound;
    private String tempReportDetails;

    public CompareListsResult() {
      aMismatchWasFound = false;
      aCommonMessageWasFound = false;
      tempReportDetails = "";
    }

    public void foundAMismatch() {
      aMismatchWasFound = true;
    }

    public void foundACommonMessage() {
      aCommonMessageWasFound = true;
    }

    public void addReportLine(String thisLine) {
      tempReportDetails += thisLine + "\n";
    }

    public boolean aMismatchWasFound() {
      return aMismatchWasFound;
    }

    public boolean aCommonMessageWasFound() {
      return aCommonMessageWasFound;
    }

    public String getReport() {
      return tempReportDetails;
    }
```

```
  }
}
```

F.4.30 Message Under Data 1

```java
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.AcceptableException;
import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.Helper;
import uk.ac.ncl.cjg.ws_enhanced.common.MessageComparison;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class MessageUnderData1 implements IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis
    Wait.delayAnalysis();

    String ruleID = null;
    String ruleIDNoMessageNumber = "ActiveAnalysisMessageUnderData1";
    IAcmeConnector theElement = null;
    IAcmePort port1 = null;
    IAcmePort port2 = null;
    Integer theMessageIndex = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeConnector) i.next();
      port1 = (IAcmePort) i.next();
      port2 = (IAcmePort) i.next();
      theMessageIndex = (Integer) i.next();
      ruleID = ruleIDNoMessageNumber + "-msg" + theMessageIndex;
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check rule is active
    try{
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(
          ruleIDNoMessageNumber, theElement)) {
        Reporter.report(theElement, ruleID, "");
```

```
        return Boolean.TRUE;
      }
    } catch (ReportableException rE){
      Reporter
      .report(
          theElement,
          ruleID,
          "There was a reportable Exception raised when getting the activity status of this analysis: \n",
          rE);
  return Boolean.FALSE;

    } catch (Exception e){
      Reporter
      .report(
          theElement,
          ruleID,
          "There was a general Exception raised when getting the activity status of this analysis: \n",
          e);
  return Boolean.FALSE;
    }

  // perform the analysis
  try {

    theResult = MessageComparison.messageUnderData1(port1,
        port2,
        theMessageIndex);


  } catch (AcceptableException e) {
    theResult = new AnalysisResult(true, "");
  } catch (ReportableException e) {
    Reporter.report(theElement, ruleID, e.getMessage());
    return Boolean.FALSE;
  } catch (Exception e) {
    Reporter
        .report(
            theElement,
            ruleID,
            "There was an Exception raised performing the analysis: \n",
            e);
    return Boolean.FALSE;
  }


  // report and return the results
  Reporter.report(theElement, ruleID, theResult.getReport());
  if (theResult.getResult() == true)
    return Boolean.TRUE;
  else
    return Boolean.FALSE;
 }

}
```

F.4.31 Message Under Data 2

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.
    IExternalAnalysisExpressionNode;
```

```java
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.AcceptableException;
import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.MessageComparison;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class MessageUnderData2 implements IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis
    Wait.delayAnalysis();

    String ruleID = null;
    String ruleIDNoMessageNumber = "ActiveAnalysisMessageUnderData2";
    IAcmeConnector theElement = null;
    IAcmePort port1 = null;
    IAcmePort port2 = null;
    Integer theMessageIndex = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeConnector) i.next();
      port1 = (IAcmePort) i.next();
      port2 = (IAcmePort) i.next();
      theMessageIndex = (Integer) i.next();
      ruleID = ruleIDNoMessageNumber + "-msg" + theMessageIndex;
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try{
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(
          ruleIDNoMessageNumber, theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
      } catch (ReportableException rE){
        Reporter
        .report(
            theElement,
            ruleID,
            "There was a reportable Exception raised when getting the activity status of this analysis: \n",
            rE);
    return Boolean.FALSE;

      } catch (Exception e){
        Reporter
        .report(
            theElement,
```

```
          ruleID,
          "There was a general Exception raised when getting
              the activity status of this analysis: \n",
          e);
    return Boolean.FALSE;
      }

    // perform the analysis
    try {
      theResult = MessageComparison.messageUnderData2(port1,
          port2,
          theMessageIndex);
    } catch (AcceptableException e) {
      theResult = new AnalysisResult(true, "");
    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the
                  analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;
  }

}
```

F.4.32 Message Vector

```
package uk.ac.ncl.cjg.ws_enhanced.common;

public class MessageVector {

  public static final String NO_MAPPING = "-1";
  public static final int NO_MAPPING_INDEX = -1;
  private int sendsFirstCSPIndex, receivesFirstCSPIndex;
  private String sentMessageID, receivedMessageID;
  private boolean fromSendsFirst;

  public MessageVector(int sendsFirstCSPIndex, int
      receivesFirstCSPIndex, boolean fromSendsFirst, String
      sendsFirstCSP, String receivesFirstCSP) throws Exception
      {
    this.sendsFirstCSPIndex = sendsFirstCSPIndex;
    this.receivesFirstCSPIndex = receivesFirstCSPIndex;
    this.fromSendsFirst = fromSendsFirst;
    // get the correct message IDs accounting for the direction
        of the message.

    String tempDebug = "Message Vector Creator: \n ";
    tempDebug+= "sends first csp index    " +
        sendsFirstCSPIndex + "\n";
    tempDebug += "receivesFirst csp index    " +
        receivesFirstCSPIndex + "\n";
    tempDebug += "sendsfirstCSP  \n " + sendsFirstCSP +"\n";
    tempDebug += "receivedfirstCSP \n" + receivesFirstCSP +"\n"
        ;


    if (fromSendsFirst)
    {
```

```
      sentMessageID = DataExtractionUtils.getMessageNameFromCSPAtLine(sendsFirstCSP, sendsFirstCSPIndex);
      receivedMessageID = DataExtractionUtils.getMessageNameFromCSPAtLine(receivesFirstCSP, receivesFirstCSPIndex);
    }
    else
    {
      receivedMessageID = DataExtractionUtils.getMessageNameFromCSPAtLine(sendsFirstCSP, sendsFirstCSPIndex);
      sentMessageID = DataExtractionUtils.getMessageNameFromCSPAtLine(receivesFirstCSP, receivesFirstCSPIndex);
    }

    tempDebug +="sentMessageID  " + sentMessageID + "\n";
    tempDebug +="receivedMessageID  " + receivedMessageID + "\n";

    Helper.writeDebug(tempDebug);

  }


  public int getSendsFirstCSPIndex() {
    return sendsFirstCSPIndex;
  }

  public int getReceivesFirstCSPIndex() {
    return receivesFirstCSPIndex;
  }

  public boolean directionIsFromSendsFirst() {
    return fromSendsFirst;
  }

  public String getSentMessageID(){
    return sentMessageID;
  }

  public String getreceivedMessageID(){
    return receivedMessageID;
  }
}
```

F.4.33 Omission Mismatch

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPConnectorConstructor;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPHidingSetConstructor;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPModelBuilder;
import uk.ac.ncl.cjg.ws_enhanced.common.FDRResultsAnalyzer;
```

```java
import uk.ac.ncl.cjg.ws_enhanced.common.Helper;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class OmissionMismatch implements
    IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws
          AcmeExpressionEvaluationException {
    // pause the analysis to allow AcmeStudio to do something
        other than
    // external analysis

    Wait.delayAnalysis();

    // extract data types from analysis call, this should be
        passed
    // a single component
    String ruleID = "ActiveAnalysisOmissionMismatch";
    IAcmeComponent theElement = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeComponent) i.next();
    } catch (Exception e) {
      Reporter.report(ruleID,
          "There was a problem extracting the required data: \n
              ", e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID
          ,
          theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when
                  getting the activity status of this analysis:
                   \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a general Exception raised when
                  getting the activity status of this analysis:
                   \n",
              e);
      return Boolean.FALSE;
    }

    // perform the analysis
    try {

```

```java

String outputPath = "/home/carl/analysisModel.csp";
String focusCompID = theElement.getName();



// perform the deadlock portion of this analysis

List fdrRawResultsDeadLock = new LinkedList<String>();
String outputPath2 = "/home/carl/output2.csp";

// themodel 0 = string csp model
// themodel 1 = CSPHidingSetConstructor
ArrayList theModelDeadLock = CSPModelBuilder.buildModel(
    CSPModelBuilder.ANALYSIS_DEADLOCK_OMISSION_SUPPORT,
        focusCompID, null, theElement);

String theCSPModelDeadLock = (String) theModelDeadLock.
    get(0);
CSPHidingSetConstructor hidCon = (CSPHidingSetConstructor
    ) theModelDeadLock
    .get(1);
CSPConnectorConstructor connCon = (
    CSPConnectorConstructor) theModelDeadLock
.get(2);
Helper.writeDebug(theCSPModelDeadLock);
Helper.writeModelToFile(theCSPModelDeadLock, outputPath);
fdrRawResultsDeadLock = Helper.processCSPModel(outputPath
    , 100);

// perform the refinement portion of this analysis

List fdrRawResultsRefinement = new LinkedList<String>();
ArrayList theModelRefinement = CSPModelBuilder.buildModel
    (
    CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT,
        focusCompID, null, theElement);

String theCSPModelRefinement = (String)
    theModelRefinement.get(0);
Helper.writeDebug(theCSPModelRefinement);
Helper.writeModelToFile(theCSPModelRefinement,
    outputPath2);
fdrRawResultsRefinement = Helper.processCSPModel(
    outputPath2, 100);

FDRResultsAnalyzer ra = new FDRResultsAnalyzer(
    CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT,
    (CSPHidingSetConstructor) hidCon, focusCompID,
        connCon);

Helper.openDebug(focusCompID +"_"+ ruleID + ".txt");

Helper.writeDebug("Deadlock raw retuls: " +
    fdrRawResultsDeadLock);
Helper.writeDebug("refinement raw retuls: " +
    fdrRawResultsRefinement);
ra.submitDeadlockTraces(fdrRawResultsDeadLock);
ra.submitRefinementTraces(fdrRawResultsRefinement);

// ra.repoart results is true if the analysis failed,
    while the analysis
// result expects a failed analysis to return false.
if(ra.reportResult())
{
  theResult = new AnalysisResult(false, ra.reportDetails
      ());
}
else
{
```

```
          theResult = new AnalysisResult(true, ra.reportDetails()
              );
        }

        Helper.closeDebug();

      } catch (ReportableException e) {
        Reporter.report(theElement, ruleID, e.getMessage());
        return Boolean.FALSE;
      } catch (Exception e) {
        Reporter
            .report(
                theElement,
                ruleID,
                "There was an Exception raised performing the
                    analysis: \n",
                e);
        return Boolean.FALSE;
      }

      // report and return the results
      Reporter.report(theElement, ruleID, theResult.getReport());
      if (theResult.getResult() == true)
        return Boolean.TRUE;
      else
        return Boolean.FALSE;

    }

}
```

F.4.34 Omission Partial Mismatch

```
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.
    IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.
    AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPConnectorConstructor
    ;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPHidingSetConstructor
    ;
import uk.ac.ncl.cjg.ws_enhanced.common.CSPModelBuilder;
import uk.ac.ncl.cjg.ws_enhanced.common.FDRResultsAnalyzer;
import uk.ac.ncl.cjg.ws_enhanced.common.Helper;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class OmissionPartialMatch implements
    IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws
          AcmeExpressionEvaluationException {

      // pause the analysis to allow AcmeStudio to do something
           other than
      // external analysis
```

```
        Wait.delayAnalysis();

        // extract data types from analysis call, this should be passed
        // a single component
        String ruleID = "ActiveAnalysisOmissionPartialMatch";
        IAcmeComponent theElement = null;
        AnalysisResult theResult = null;

        java.util.Iterator i = arg1.iterator();

        // extract the required model elements from the passed list
        try {
          theElement = (IAcmeComponent) i.next();
        } catch (Exception e) {
          Reporter.report(ruleID,
              "There was a problem extracting the required data: \n", e);
          return Boolean.FALSE;
        }

        // check if this rule is active
        try {
          if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(
              ruleID,
              theElement)) {
            Reporter.report(theElement, ruleID, "");
            return Boolean.TRUE;
          }
        } catch (ReportableException rE) {
          Reporter
              .report(
                  theElement,
                  ruleID,
                  "There was a reportable Exception raised when getting the activity status of this analysis: \n",
                  rE);
          return Boolean.FALSE;

        } catch (Exception e) {
          Reporter
              .report(
                  theElement,
                  ruleID,
                  "There was a general Exception raised when getting the activity status of this analysis: \n",
                  e);
          return Boolean.FALSE;
        }

        // perform the analysis
        try {

          String outputPath = "/home/carl/analysisModel.csp";
          String focusCompID = theElement.getName();



          // perform the deadlock portion of this analysis

          List fdrRawResultsDeadLock = new LinkedList<String>();
          String outputPath2 = "/home/carl/output2.csp";

          // themodel 0 = string csp model
          // themodel 1 = CSPHidingSetConstructor
          ArrayList theModelDeadLock = CSPModelBuilder.buildModel
              (
```

```java
            CSPModelBuilder.ANALYSIS_DEADLOCK_OMISSION_SUPPORT,
                focusCompID, null, theElement);

        String theCSPModelDeadLock = (String) theModelDeadLock.
            get(0);
        CSPHidingSetConstructor hidCon = (
            CSPHidingSetConstructor) theModelDeadLock
            .get(1);
        CSPConnectorConstructor connCon = (
            CSPConnectorConstructor) theModelDeadLock
        .get(2);

        Helper.writeDebug(theCSPModelDeadLock);

        Helper.writeModelToFile(theCSPModelDeadLock, outputPath
            );
        fdrRawResultsDeadLock = Helper.processCSPModel(
            outputPath, 100);

        // perform the refinement portion of this analysis

        List fdrRawResultsRefinement = new LinkedList<String>()
            ;
        ArrayList theModelRefinement = CSPModelBuilder.
            buildModel(
            CSPModelBuilder.ANALYSIS_COMPONENT_REFINEMENT,
                focusCompID, null, theElement);

        String theCSPModelRefinement = (String)
            theModelRefinement.get(0);

        Helper.writeDebug(theCSPModelRefinement);
        Helper.writeModelToFile(theCSPModelRefinement,
            outputPath2);
        fdrRawResultsRefinement = Helper.processCSPModel(
            outputPath2, 100);

        FDRResultsAnalyzer ra = new FDRResultsAnalyzer(
            CSPModelBuilder.
            ANALYSIS_COMPONENT_REFINEMENT_PARTIAL,
            (CSPHidingSetConstructor) hidCon, focusCompID,
                connCon);


        //Helper.openDebug(focusCompID +"_"+ ruleID + ".txt");
        Helper.writeDebug("Deadlock raw retuls: " +
            fdrRawResultsDeadLock);
        Helper.writeDebug("refinement raw retuls: " +
            fdrRawResultsRefinement);


        ra.submitDeadlockTraces(fdrRawResultsDeadLock);
        ra.submitRefinementTraces(fdrRawResultsRefinement);

        // ra.repoart results is true if the analysis failed,
            while the analysis
        // result expects a failed analysis to return false.


        if(ra.reportResult())
        {
          theResult = new AnalysisResult(false, ra.
              reportDetails());
        }
        else
        {
          theResult = new AnalysisResult(true, ra.reportDetails
              ());
        }

        //Helper.closeDebug();
```

```
      } catch (ReportableException e) {
        Reporter.report(theElement, ruleID, e.getMessage());
        return Boolean.FALSE;
      } catch (Exception e) {
        Reporter
            .report(
                theElement,
                ruleID,
                "There was an Exception raised performing the analysis: \n",
                e);
        return Boolean.FALSE;
      }

      // report and return the results
      Reporter.report(theElement, ruleID, theResult.getReport());
      if (theResult.getResult() == true)
        return Boolean.TRUE;
      else
        return Boolean.FALSE;

  }

}
```

F.4.35 Port

```
package uk.ac.ncl.cjg.ws_enhanced.common;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;


public class Port implements Comparable<Port> {

  public String iD;
  public String messagePattern;
  public boolean reentrant;
  public boolean isUnicast;
  public String choiceGroup;
  public boolean choiceGroupMaker;
  public boolean inOurControlDomain;
  public Set attachedTo;
  public Component childOf;
  public Map messages = new TreeMap();

  public Port(String iD)
  {
    this.iD = iD;
    attachedTo = new TreeSet();
  }

  public int compareTo(Port other)
  {
    Port otherPort = (Port)other;
    return this.iD.compareTo(otherPort.iD);
  }

  public String toString()
  {
    String toReturn = "";

    toReturn += "/n /n port id \n";
    toReturn += iD;
    toReturn += "       messagePatterb \n";
    toReturn += "" + messagePattern + " \n";
    toReturn += "       reentrant \n";
    toReturn += "" +reentrant  + " \n";
    toReturn += "       is unicast \n";
```

```
    toReturn += "" + isUnicast + " \n";
    toReturn += "       choice group \n";
    toReturn += "" + choiceGroup + " \n";
    toReturn += "       choicegroupmaker \n";
    toReturn += "" + choiceGroupMaker + " \n";
    toReturn += "       in our control domian \n";
    toReturn += "" + inOurControlDomain + " \n";
    toReturn += " attached to " + attachedTo.size() + " \n";

    Iterator i = attachedTo.iterator();
    while(i.hasNext())
    {
      Connector c = (Connector)i.next();
      toReturn += " conn : " + c.iD + " \n";
    }

    return toReturn;
  }

}
```

F.4.36 Reportable Exception

```
package uk.ac.ncl.cjg.ws_enhanced.common;

public class ReportableException extends Exception {
  public ReportableException()
  {
    super();
  }

  public ReportableException(String message)
  {
    super(message);
  }

  public ReportableException(String message, Throwable cause)
  {
    super(message, cause);
  }

  public ReportableException(Throwable cause)
  {
    super(cause);
  }
}
```

F.4.37 Reporter

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import java.io.File;
import java.io.PrintWriter;
import java.util.Iterator;
import java.util.Set;

import org.acmestudio.acme.core.type.IAcmeStringValue;
import org.acmestudio.acme.element.IAcmeComponent;
import org.acmestudio.acme.element.IAcmeElement;
import org.acmestudio.acme.element.IAcmeSystem;
import org.acmestudio.acme.element.property.IAcmeProperty;
import org.acmestudio.acme.element.property.IAcmePropertyValue;

public class Reporter {

  /**
   * Outputs the report to a file named <qualifiedElementName
       >.<RuleName>.txt
   * The file will be placed in the path described in the
       analysis control
   * element.
   *
```

```
 * If the report is the empty String "" then the file will be
      deleted if it
 * exists.
 *
 * @param theElement
 *            The architectural element from which the rule
     was invoked
 * @param theReporte
 *            The report to be dumped, unceremoniously into
     the text file
 */

private static final String BASE_PATH = "/home/carl/
    acmeOutput/";

public static void report(IAcmeElement theElement, String
    ruleID,
    String theReport) {

  String elementID = theElement.getQualifiedName();
  //String fullOutputPath = getOutputPath(theElement) +
      elementID + "."
  String fullOutputPath = BASE_PATH + elementID + "."
      + ruleID + ".txt";
  outputReport(fullOutputPath, theReport);
}

public static void report(String ruleID, String theReport) {
  String fullOutputPath = BASE_PATH + ruleID + ".txt";
  outputReport(fullOutputPath, theReport);
}

public static void report(IAcmeElement theElement, String
    ruleID,
    String reportNote, Exception theException) {

  String elementID = theElement.getQualifiedName();
  // String fullOutputPath = getOutputPath(theElement) +
      elementID + "."
  String fullOutputPath = BASE_PATH + elementID + "."
      + ruleID + ".Exception.txt";
  String theReport = reportNote;
  theReport += getExceptionDetails(theException);
  // StackTraceElement[] theStackTrace = theException.
      getStackTrace();
  // for (int i=0; i<theStackTrace.length; i++)
  // {
  // theReport += "\n " + theStackTrace[i];
  // }
  outputReport(fullOutputPath, theReport);
}

public static void report(String ruleID, String reportNote,
    Exception theException) {
  String fullOutputPath = BASE_PATH + ruleID + ".Exception.
      txt";
  String theReport = reportNote;
  theReport += getExceptionDetails(theException);
  // StackTraceElement[] theStackTrace = theException.
      getStackTrace();
  // for (int i=0; i<theStackTrace.length; i++)
  // {
  // theReport += "\n " + theStackTrace[i];
  // }
  outputReport(fullOutputPath, theReport);
}

private static String getExceptionDetails(Exception
    theException) {
  String theDetails = "";
  theDetails += "The Cause : " + theException.getCause() + "
      \n";
```

```
    theDetails += "toString : " + theException + " \n";

    StackTraceElement[] theStackTrace = theException.
        getStackTrace();
    for (int i = 0; i < theStackTrace.length; i++) {
      theDetails += "\n " + theStackTrace[i];
    }

    return theDetails;
  }

  private static void outputReport(String thePath, String
      theReport) {
    if (theReport.equalsIgnoreCase("")) {
      File fileToDelete = new File(thePath);
      try {
        fileToDelete.delete();
      } catch (Exception e) {
        try {
          PrintWriter pw = new PrintWriter(
              BASE_PATH + "reporter-report-exception.txt");
          pw.println(e);
          pw.flush();
        } catch (Exception ee) {
        }
      }
    } else {
      try {
        PrintWriter pw = new PrintWriter(thePath);
        pw.println(theReport);
        pw.flush();
      } catch (Exception e) {
        try {
          PrintWriter pw = new PrintWriter(
              BASE_PATH + "reporter-report-exception.txt");
          pw.println(e);
          pw.flush();
        } catch (Exception ee) {
        }
      }
    }
  }

  private static String getOutputPath(IAcmeElement
      elementRuleIsIn) {
    try {
      final String analysisControllerType = "
          CompTWSAnalysisControl";

      // move up the tree till we get the IAcmeSystem object
      IAcmeElement theParent = elementRuleIsIn.getParent();
      IAcmeSystem theSystem = null;

      while (!(theParent instanceof IAcmeSystem)) {
        theParent = theParent.getParent();
        if (theParent == null || !(theParent instanceof
            IAcmeElement))
          return "BASE_PATH";
      }
      theSystem = (IAcmeSystem) theParent;

      // get the list of all components in that system
      // move through the list till we find one of the correct
          type
      IAcmeComponent theAnalysisController = null;
      Set theComponents = theSystem.getComponents(); // maybe
          should
                                   // parameterize the
                                   // set here

      Iterator i = theComponents.iterator();
      while (i.hasNext()) {
```

```java
        IAcmeComponent thisComponent = (IAcmeComponent) i.next();
        if (thisComponent.declaresType(analysisControllerType)) {
          theAnalysisController = thisComponent;
          break;
        }
      }

      if (theAnalysisController == null)
        return "BASE_PATH";

      // move through all properties of the component to find the one we
      // are looking for
      // and return its value.

      IAcmeProperty analysisActiveProperty = theAnalysisController
          .getProperty("outputPath");
      if (analysisActiveProperty == null)
        return "BASE_PATH";
      IAcmePropertyValue analysisActivePropertyValue = analysisActiveProperty
          .getValue();

      if (analysisActivePropertyValue instanceof IAcmeStringValue)
        return ((IAcmeStringValue) analysisActivePropertyValue)
            .getValue()
            + "/";

      // the default action will return an empty path.
      return "BASE_PATH";
    } catch (Exception e) {
      // something went wrong, return an empty string to allow something
      // to be output, this should be replaced with a different default
      // suitable for whoever is using this stuff
      return BASE_PATH;
    }
  }

}
```

F.4.38 State Scopes Comparison

```java
package uk.ac.ncl.cjg.ws_enhanced.common;

public class StateScopeComparison {

  public static boolean exhibitedCompatibleWithExpected(String exhibited, String expected )
  {
    // define the compatible matches, all other combinations are considered incompatible

    if(expected.equalsIgnoreCase("NoPreference"))
      return true;

    if(expected.equalsIgnoreCase(exhibited))
      return true;

    return false;
  }

}
```

F.4.39 State Scopes Match

```java
package uk.ac.ncl.cjg.ws_enhanced;

import java.util.List;
import java.util.Stack;

import org.acmestudio.acme.core.IAcmeType;
import org.acmestudio.acme.element.IAcmeConnector;
import org.acmestudio.acme.element.IAcmePort;
import org.acmestudio.acme.environment.error.AcmeError;
import org.acmestudio.acme.rule.node.IExternalAnalysisExpressionNode;
import org.acmestudio.acme.rule.node.feedback.AcmeExpressionEvaluationException;

import uk.ac.ncl.cjg.ws_enhanced.common.ActiveAnalysisChecker;
import uk.ac.ncl.cjg.ws_enhanced.common.AnalysisResult;
import uk.ac.ncl.cjg.ws_enhanced.common.MessageComparison;
import uk.ac.ncl.cjg.ws_enhanced.common.ReportableException;
import uk.ac.ncl.cjg.ws_enhanced.common.Reporter;
import uk.ac.ncl.cjg.ws_enhanced.common.Wait;

public class StateScopesMatch implements IExternalAnalysisExpressionNode {

  @Override
  public Object evaluate(IAcmeType arg0, List<Object> arg1,
      Stack<AcmeError> arg2) throws AcmeExpressionEvaluationException {

    // pause the analysis to allow AcmeStudio to do something other than
    // external analysis

    Wait.delayAnalysis();

    // extract data types from analysis call, this should be passed
    // a single component
    String ruleID = "ActiveAnalysisCentralDataStoreCorrect";
    IAcmeConnector theElement = null;
    IAcmePort port1 = null;
    IAcmePort port2 = null;
    AnalysisResult theResult = null;

    java.util.Iterator i = arg1.iterator();

    // extract the required model elements from the passed list
    try {
      theElement = (IAcmeConnector) i.next();
      port1 = (IAcmePort) i.next();
      port2 = (IAcmePort) i.next();
    } catch (Exception e) {
      Reporter
          .report(
              ruleID,
              "Some fo the required elements required "
              +"(the connector and both attached ports) were"
              + "not passed by acme to the analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // check if this rule is active
    try {
      if (!ActiveAnalysisChecker.CheckIfAnalysisIsActive(ruleID
          , theElement)) {
        Reporter.report(theElement, ruleID, "");
        return Boolean.TRUE;
      }
    } catch (ReportableException rE) {
```

```
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a reportable Exception raised when getting the activity status of this analysis: \n",
              rE);
      return Boolean.FALSE;

    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was a general Exception raised when getting the activity status of this analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // perform the analysis
    try {

      theResult = MessageComparison.stateScopesMatch(theElement, port1, port2);

    } catch (ReportableException e) {
      Reporter.report(theElement, ruleID, e.getMessage());
      return Boolean.FALSE;
    } catch (Exception e) {
      Reporter
          .report(
              theElement,
              ruleID,
              "There was an Exception raised performing the analysis: \n",
              e);
      return Boolean.FALSE;
    }

    // report and return the results
    Reporter.report(theElement, ruleID, theResult.getReport());
    if (theResult.getResult() == true)
      return Boolean.TRUE;
    else
      return Boolean.FALSE;

  }
}
```

F.4.40 T Data Rep

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import org.acmestudio.acme.core.type.IAcmeEnumValue;
import org.acmestudio.acme.element.property.IAcmePropertyValue;

public class TDataRep implements Comparable<TDataRep>{

  private String theValue;

  public TDataRep (IAcmePropertyValue propertyFromStyle)
  {
    theValue = ((IAcmeEnumValue)propertyFromStyle).getValue();
  }

  public int compareTo(TDataRep theOther)
  {
    // there is no natural order to these enumerated values, so the
```

```
    // natural string value will be used

    return theValue.compareTo(theOther.theValue);
  }

  public boolean compatibleWith(TDataRep theOther)
  {
    if (this.compareTo(theOther) == 0) return true;
    else return false;
  }

  public String toString()
  {
    return theValue;
  }

}
```

F.4.41 T Data Semantics

```
package uk.ac.ncl.cjg.ws_enhanced.common;

import org.acmestudio.acme.core.type.IAcmeStringValue;
import org.acmestudio.acme.element.property.IAcmePropertyValue;

public class TDataSemantics implements Comparable<
    TDataSemantics>{

  private String theSemantics;

  public TDataSemantics(IAcmePropertyValue theSemantics)
  {
    // The semantics are simply a string in this version
    this.theSemantics = ((IAcmeStringValue)theSemantics).
        getValue();
  }

  public String toString(){
    return "The semantics are : " + theSemantics;
  }

  public int compareTo(TDataSemantics theOther)
  {
    return theSemantics.compareToIgnoreCase(theOther.
        theSemantics);
  }

  public boolean compatibleWith(TDataSemantics theOther)
  {
    if (this.compareTo(theOther)==0) return true;
    else return false;
  }
}
```

F.4.42 T Safe Boolean

```
package uk.ac.ncl.cjg.ws_enhanced.common;

public class TSafeBoolean {

  public static final int UNDEFINED = 0;
  public static final int YES = 1;
  public static final int NO = 2;

  private int state;

  public TSafeBoolean(int theState )
  {
    state = theState;
  }

  public boolean hasState(int stateToCheckFor)
```

```
  {
    return state == stateToCheckFor;
  }
}
```

F.4.43 Wait

```
package uk.ac.ncl.cjg.ws_enhanced.common;

/**
 *
 * @author carl, extended from the Wait class that can be found
       at java-tips.org
 *
 * http://www.java-tips.org/java-se-tips/java.lang/pause-the-
       execution.html
 */

public class Wait {
  public static void oneSec() {
    try {
      Thread.currentThread().sleep(1000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

  public static void manySec(long s) {
    try {
      Thread.currentThread().sleep(s * 1000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

  public static void delayAnalysis() {
    try {
      Thread.currentThread().sleep(100);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

}
```

Appendix G

Traces Tables

This appendix presents the complete set of message traces for the "sensible" combinations of port types, i.e. those where one port expects to send the first message (outbound) while the other port expects to receive the first message (inbound). These include both the natural combinations of ports, such as *out-only* to *in-only* but also those combinations where the are only a few or no individual traces in common.

There are four graphs included, each focussing on a single type of outbound port and representing its interactions with the four inbound port types. The purpose of the tables are to indicate, for each pairing of port types, if they have any common traces and also if they have any divergent traces. A divergent trace, labelled 'Dx', is one where the expectations of one of the ports exceeds the expectations of the other. This may be in terms of sending unexpected messages, commission, or expecting non-existent messages, omission.

This data is used as a look-up by the two external analysis classes concerned with detecting mismatch between the message exchange patterns of two connected ports

Here is a legend of the symbols used to represent the messages (*Msg.*) and origins (*Orig.*) in the tables:

- Message names

  req The initial message sent in the pattern, termed here the request though the term 'notification' may be more apt in the cases where no response is expected.

  res The response message to the request.

  flt A fault message generated in response to the request.

  flt2 A fault message generated as a result of the response to the earlier request.

- Origins

  ob 'outbound', this is a message that will be sent from the outbound port at this point in the trace

ib 'inbound', this is a message that will be sent from the inbound port at this point in the trace

obd 'outbound desires', this is a message that the outbound port would like to send at this point in the trace, this message is not expected by the inbound port and so is not allowed

ibd 'inbound desires', this is a message that the inbound port would like to send at this point in the trace, this is a message that is not expected by the outbound port and so is not allowed

obdi 'outbound desires inbound', this is a message that the outbound port desires that the inbound port sends to it at this point in the trace, this message is not included in the inbound port's message exchange pattern and so will not be sent

ibdo 'inbound desires outbound', this is a message that the inbound port desires that the outbound port sends to it at this point in the trace, this message is not included in the outbound port's message exchange pattern and so will not be sent

	Traces of Notification with ...					
ID	*Msg.*	*Orig.*	*Msg.*	*Orig.*	*Msg.*	*Org.*
In-only						
T1	req	ob				
Robust-in-only						
T1	req	ob				
D1	req	ob	flt	ibd		
Request-response						
D2	req	ob	res	ibd		
D3	req	ob	flt	ibd		
In-optional-out						
T1	req	ob				
D1	req	ob	res	ibd		
D2	req	ob	flt	ibd		
D3	req	ob	res	ibd	flt2	ibdo

Table G.1: The traces between a notification port and all four inbound port types.

	Traces of Robust-out-only with ...					
ID	*Msg.*	*Orig.*	*Msg.*	*Orig.*	*Msg.*	*Org.*
In-only						
T1	req	ob				
D1	req	ob	flt	ibd		
Robust-in-only						
T1	req	ob				
T2	req	ob	flt	ib		
Request-response						
D2	req	ob	res	ibd		
T1	req	ob	flt	ib		
In-optional-out						
T1	req	ob				
D1	req	ob	res	ibd		
T2	req	ob	flt	ib		
D2	req	ob	res	ibd	flt2	ibdo

Table G.2: The traces between a robust-out-only port and all four inbound port types.

	Traces of Solicit-response with ...					
ID	*Msg.*	*Orig.*	*Msg.*	*Orig.*	*Msg.*	*Org.*
In-only						
D1	req	ob	res	ibd		
D2	req	ob	res	ibd		
Robust-in-only						
D1	req	ob	res	ibd		
T1	req	ob	flt	ib		
Request-response						
T1	req	ob	res	ib		
T2	req	ob	flt	ib		
In-optional-out						
T1	req	ob	res	ib		
T2	req	ob	flt	ib		
D2	req	ob	res	ib	flt2	ibdo

Table G.3: The traces between a solicit-response port and all four inbound port types.

	Traces of Out-optional-in with ...					
ID	*Msg.*	*Orig.*	*Msg.*	*Orig.*	*Msg.*	*Org.*
In-only						
T1	req	ob				
D1	req	ob	res	obdi		
D2	req	ob	flt	obdi		
D3	req	ob	res	obdi	flt2	obd
Robust-in-only						
T1	req	ob				
D1	req	ob	res	obdi		
T2	req	ob	flt	ib		
D2	req	ob	res	obdi	flt2	obd
Request-response						
T1	req	ob	res	ib		
T2	req	ob	flt	ib		
D1	req	ob	res	ib	flt2	obd
In-optional-out						
T1	req	ob				
T2	req	ob	res	ib		
T3	req	ob	flt	ib		
T4	req	ob	res	ib	flt2	ob

Table G.4: The traces between an out-optional-in port and all four inbound port types.

Appendix H

CSP Introduction

The formal process algebra CSP (Communicating Sequential Processes) is used in this work:

- to represent the message passing choreography expected by an individual component; and
- to assess the resulting system for certain types of mismatch.

This appendix gives a non-formal introduction to the parts of CSP that are employed by the style. This is not intended to be a tutorial as such, but should afford the reader sufficient appreciation of the syntax and meaning to be able to understand the CSP described.

H.1 Model Definition

H.1.1 Linear Process Definition

The most basic CSP concept is the process and the most simple processes are linear sequences of events. For example:

$$PROC\_A \widehat{=} a \rightarrow b \rightarrow Stop$$

This defines a process called $PROC\_A$ that performs an event a then performs an event b and finally acts like the special CSP process $STOP$. The stop process is predefined in CSP and it refuses to perform any events.

Processes can call other user defined processes as well:

$$PROC\_B \widehat{=} a \rightarrow b \rightarrow PROC\_C$$
$$PROC\_C \widehat{=} c \rightarrow Stop$$

Here when the events a and b have been performed $PROC\_A$ then acts like $PROC\_C$ and performs event c before stopping. The names of the processes are not significant other than being identifiers and the fact that the system is defined using two processes is also not significant. The following process behaves identically:

$$PROC\_D \mathrel{\widehat{=}} a \rightarrow b \rightarrow c \rightarrow Stop$$

H.1.2 Concurrency

CSP supports the definition of systems with multiple concurrently executing processes. There are two constructs to represent this, parallel and interleaved.

H.1.2.1 Interleaved

The simplest form of concurrency is interleaved. Here a processes is defined as acting like two or more other processes and those processes are independent of each other. For example:

$$PROC\_E \mathrel{\widehat{=}} a \rightarrow b \rightarrow Stop$$
$$PROC\_F \mathrel{\widehat{=}} b \rightarrow c \rightarrow Stop$$
$$PROC\_INTERLEAVE \mathrel{\widehat{=}} PROC\_E \mathbin{|||} PROC\_F$$

The process $PROC\_INTERLEAVE$ acts as if both $PROC\_A$ and $PROC\_B$ are running but each each step in the execution only one of them can perform an event. This means an observer of the system would witness the following execution traces:

1. a,b,b,c
2. a,b,c,b
3. b,a,b,c
4. b,c,a,b

H.1.2.2 Alphabetised Parallel

The second form on concurrency is alphabetized parallel. In this case a process is defined as acting like two others but unlike interleaved, where the processes are independent, here the two processes can be forced to synchronise on any events in either of their alphabets. Synchronising means that each event in the synchronisation set must be performed by both processes simultaneously. If one of the processes reaches a point where it must next perform an event in the synchronisation set then it will be blocked from performing that event until the other process is also willing to perform it.

The previous example system can be altered to be parallel and synchronised on the b event instead of interleaved as follows:

$$PROC\_E \mathrel{\widehat{=}} a \rightarrow b \rightarrow Stop$$
$$PROC\_F \mathrel{\widehat{=}} b \rightarrow c \rightarrow Stop$$
$$PROC\_PARALLEL \mathrel{\widehat{=}} PROC\_E \mathbin{|[\, b \,]|} PROC\_F$$

As before the *PROC_PARALLEL* process acts as if both *PROC_A* and *PROC_B* are running except that in this case both processes must partake in any **b** events, so an observer of the system would see only a single trace for this system:

1. a,b,c

In the previous interleaved case there were four unique traces, now there is only a single trace. The reasons for this are entirely because of the need to synchronise on the b event. This means that the *PROC_B* process can no longer perform the first event in the system trace as it must wait for *PROC_A* to be also willing to perform it, so traces 3 and 4 above are not possible. Then once the b event has been performed *PROC_A* can only perform *Stop* leaving *PROC_B* to perform c. The result is that trace 2 from the interleaving is not possible and leaves the single trace possible with this system.

H.1.3 Process Branching

The above represents very basic processes that include no choice in the sequence of events to perform. CSP has two mechanisms to introduce branching into a process, these are internal choice and external choice. Both mechanisms allow the description of a process that, for example, performs event a then either performs b or c. Both operators are shown below, internal choice first and then external choice:

$$\begin{aligned} INT\_CHOICE \mathrel{\widehat{=}} a \rightarrow (&b \rightarrow Stop \\ &\sqcap\ c \rightarrow Stop) \end{aligned}$$

$$\begin{aligned} EXT\_CHOICE \mathrel{\widehat{=}} a \rightarrow (&b \rightarrow Stop \\ &\Box\ c \rightarrow Stop) \end{aligned}$$

The difference between them lies in whether the first event on a branch is considered before a branch is selected or not.

The internal mechanism does not consider the first event on a branch before the branch is chosen. The result of this is that the process may then follow a branch where the system is not willing to perform the first event at that point in the trace. For example:

$$
\begin{aligned}
&PROC\_OTHER \mathrel{\widehat=} a \rightarrow c \rightarrow Stop \\
&INT\_CHOICE \mathrel{\widehat=} a \rightarrow (b \rightarrow Stop \\
&\qquad\qquad\qquad\qquad \sqcap\ c \rightarrow Stop) \\
&SYSTEM\_INT \mathrel{\widehat=} PROC\_OTHER \,|[\, a, b, c \,]|\, INT\_CHOICE
\end{aligned}
$$

The result of this composition are two traces:

1. a, c
2. a

In the first trace the internal choice happened to select a branch that performed the c event expected by the other process and so the trace continued to the end. However in the second trace the internal choice selected the other branch and attempted to perform a b event, this could not happen as the parallel process could only perform a c and so the process deadlocked where deadlocking is described later.

If, on the other hand, a system was produced using the external choice process as below:

$$
\begin{aligned}
&PROC\_OTHER \mathrel{\widehat=} a \rightarrow c \rightarrow Stop \\
&EXT\_CHOICE \mathrel{\widehat=} a \rightarrow (b \rightarrow Stop \\
&\qquad\qquad\qquad\qquad \Box\ c \rightarrow Stop) \\
&SYSTEM\_EXT \mathrel{\widehat=} PROC\_OTHER \,|[\, a, b, c \,]|\, EXT\_CHOICE
\end{aligned}
$$

Then only a single trace would be observed:

1. a, c

Here, only the branch that leads to an event that the system is willing to allow is followed and so the system does not deadlock.

The external choice could then be described as being cooperative as it only attempts to perform an event that the system will allow. For the purpose of the analysis in this work, external choice is used exclusively as this allows the branch choices to be influenced by the environment and in doing so allows exploration of all possible branches of the conversation tree of a system.

H.2 Model Analysis

H.2.1 Deadlock

Deadlock is a concept that is not unique to CSP. A system is said to be deadlocked if all processes are waiting for some other process to perform some operation and as such the system makes no progress. An example of a system that deadlocks is represented by the following CSP:

$$PROC\_A \mathrel{\widehat=} a \rightarrow c \rightarrow b \rightarrow Stop$$
$$PROC\_B \mathrel{\widehat=} b \rightarrow c \rightarrow Stop$$
$$PROC\_PARALLEL \mathrel{\widehat=} PROC\_A \,|[\, b, c \,]|\, PROC\_B$$

In this system *PROC_B* initially wants to perform event b, but as this is part of the synchronisation set it cannot until *PROC_A* is willing to. Because of this the only event that can initially occur is a performed by *PROC_A*. This leaves *PROC_A* wanting to perform event c, however c is part of the synchronisation set and so *PROC_A* must wait until *PROC_B* is also willing to perform it. At this point the system is said to be deadlocked as both processes are waiting for the other to be willing to perform different events and so the system will never progress.

The FDR model checker can be instructed to determine if a model can enter a deadlock state using the following assertion:

```
assert PROC_PARALLEL [deadlock free[F]]
```

Placing this statement into the above model would return a "false" result, showing that the system is not deadlock free and providing the trace including all events performed that lead to the deadlock condition, which in this case is simply the event a.

H.2.2 Traces Refinement

An assertion about traces refinement in CSP is a assertion about whether the traces of one process are subset-equal to the traces of a specification.

For example if a specification process is defined as:

$$\begin{aligned} SPEC \mathrel{\widehat=}\ & a \rightarrow SPEC \\ & \Box\ b \rightarrow SPEC \end{aligned}$$

Then the specification includes all traces that only include as and bs.

If the following process and assertion are then considered:

$$PROC\_0 \mathrel{\widehat=} a \rightarrow PROC\_0$$
$$SPEC \sqsubseteq \mathcal{M}_{UT} PROC\_0$$

The result of the assertion is true as as *PROC_0* defines a trace consisting wholly of as.

If the following process and assertions are then added:

$$PROC\_1 \mathrel{\widehat{=}} a \rightarrow b \rightarrow c \rightarrow PROC\_1$$
$$SPEC \sqsubseteq \mathcal{M}_{UT} PROC\_1$$
$$SPEC \sqsubseteq \mathcal{M}_{UT} PROC\_1 \setminus c$$

Then the first of the two assertions would be false as *PROC_1* contains the even c in its traces. The second assertion, however, would be true as the c events produced by the process are now hidden and so the resulting trace is identical to one of the specification traces.

The important point is that one process is a refinement of another so long as all its traces exist in the set of traces defined by the specification. Thus refinement does not guarantee that two processes are equal. To show equality of traces between two processes a two way refinement is required. For example, given two processes *P_1* and *P_2* traces equality can be demonstrated by the following two assertions being true:

$$P\_1 \sqsubseteq \mathcal{M}_{UT} P\_2$$
$$P\_2 \sqsubseteq \mathcal{M}_{UT} P\_1$$

H.3 Summary

The above summarises all the CSP principles used in this work, however a more complete CSP description may be found in Schneider'[Sch00].

Appendix I

CSP Templates

This appendix is split into three sections. The first presents the derivation of the port CSP templates from the natural language W3C descriptions. The second section shows the principles behind altering the port CSP templates to link ports together to form simple conversation flows. The final section shows templates for the central CSP and shows how these templates can be used in concert with the port CSP templates to allow multi-threading, branching and looping type conversation flows.

I.1 Port CSP Templates

This first section shows the derivation CSP templates that are employed in the `messagePattern` property of each port.

The derivations start with the textual description provided by the W3C [W3C06c, W3C06f]. From this a simple graphical representation is shown to make the interpretation of the pattern explicit. Then the refinement of the templates takes place, starting with a specification that matches graphical view and adding detail until the final templates are shown.

The final part of each derivation shows the assertions that were made to demonstrate that a composition of a pair of templates behaves identically to the specification.

For completeness the FDR version of the templates are also included to show the syntax that would actually be employed by the user of the style.

I.1.1 Notification - One-way

I.1.1.1 Template Derivation

Starting with the simplest patterns, here are the W3C description of the notification message exchange pattern:

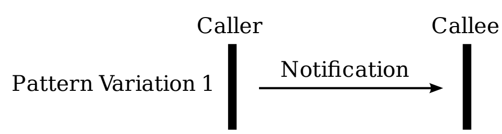
1. A message:

- indicated by a Interface Message Reference component whose message label is “out” and direction is “out”
- sent to some node N

The logical pair to the notification pattern is the in-only pattern, its text description is given below:

1. A message:
   - indicated by a Interface Message Reference component whose message label is “In” and direction is “in”
   - received from some node N

This pattern consists of only a single message, shown graphically below:



A simple specification that represents this behaviour is as follows:

$$SIMPLE\_SPEC\_NOTI \mathrel{\widehat=} Notification \rightarrow Stop$$

The simple specification can then be expanded to include the separate ‘send message’ and ‘receive message’ events that the analysis requires. This results in the following specification.

$$SPEC\_NOTI \mathrel{\widehat=} sendNote \rightarrow getNote \rightarrow Stop$$

The events relevant to the two port types are then teased apart to leave a template for the notification pattern ($NOTI$) and then one for the in-only pattern (INO):

$$NOTI \mathrel{\widehat=} sendNote \rightarrow NOTI\_OK$$
$$NOTI\_OK \mathrel{\widehat=} Stop$$

$$INO \mathrel{\widehat=} getNote \rightarrow INO\_OK$$
$$INO\_OK \mathrel{\widehat=} Stop$$

The notification / in-only message patterns are one of those where it is required to add in a faux event to indicate the point at which the conversational thread leaves the port[1]. This is done by adding a *decThread* event after the sending / receiving of the initial message as follows

$$NOTI \mathrel{\widehat=} sendNote \rightarrow NOTI\_p1$$
$$NOTI\_p1 \mathrel{\widehat=} decThread \rightarrow NOTI\_OK$$
$$NOTI\_OK \mathrel{\widehat=} Stop$$

$$INO \mathrel{\widehat=} getNote \rightarrow INO\_p1$$
$$INO\_p1 \mathrel{\widehat=} decThread \rightarrow INO\_OK$$
$$INO\_OK \mathrel{\widehat=} Stop$$

To demonstrate that a composition of the templates behaves identically to the specification, the following system is constructed and model checked. Note that because the faux *decThread* events have been added to the final templates, these must be hidden to show that the traces are identical.

$$\alpha NOTI = sendNote$$
$$\alpha INO = getNote$$
$$PORTS\_NOTI \mathrel{\widehat=} NOTI \mathbin{|||} INO$$
$$CONN\_NOTI \mathrel{\widehat=} sendNote \rightarrow getNote \rightarrow CONN\_NOTI$$
$$COMPOSED\_NOTI \mathrel{\widehat=} PORTS\_NOTI \mathbin{|[\, \alpha NOTI, \alpha INO \,]|} CONN\_NOTI$$
$$COMPOSED\_NOTI \setminus decThread \sqsubseteq \mathcal{M}_{UT} SPEC\_NOTI$$
$$SPEC\_NOTI \sqsubseteq \mathcal{M}_{UT} COMPOSED\_NOTI \setminus decThread$$

Both assertions in the above model are found to be true and in so doing show that the traces of the system composed of the two templates and connector are identical to those of the specification.

I.1.1.2 Actual Templates and Useage

When used in ACME Studio the templates differ from those presented above in that they require the name of the pattern to be included on the first line. The actual templates to be used then are as follows along with a guide to the lines and how they can be modified.

```
noti
NOTI = sendNote  -> NOTI_p1
NOTI_p1 = decThread -> NOTI_OK
NOTI_OK = STOP
```

[1]Further details regarding the need for these extra faux events may be found in Section 5.2.2.9 on page 112.

1. Line 1 contains the template ID, this should not be altered;
2. All instances of “NOTI”, except that one line 1, should be replaced with the “<componentID>-<portID>”.
3. “sendNote” should be replaced with the ID of the message this port sends;
4. “STOP” should be replaced with the ID where the process flows after this port.

```
1 ino
2 INO = getNote -> INO_p1
3 INO_p1 = decThread -> INO_OK
4 INO_OK = STOP
```

1. Line 1 contains the template ID, this should not be altered;
2. All instances of “INO”, except that one line 1, should be replaced with the “<componentID>-<portID>”.
3. “getNote” should be replaced with the ID of the message this port receives;
4. “STOP” should be replaced with the ID where the process flows after this port.

I.1.2 Robust-out-only - Robust-in-only

I.1.2.1 Template Derivation

The second pair of message exchange patterns presented are the robust-out-only / robust-in-only pairing. The W3C description of the robust-out-only message exchange pattern is as follows:

1. A message:
   - indicated by a Interface Message Reference component whose message label is “Out” and direction is “out”
   - sent to some node N

   ⋮ ⋮

   Any message, including the first in the pattern, MAY trigger a fault message, which MUST have opposite direction.

The W3C description of the matching robust-in-only pattern is as follows:

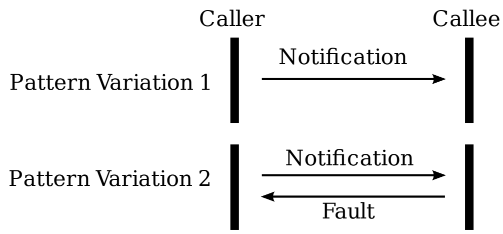
1. A message:
   - indicated by a Interface Message Reference component whose message label is “In” and direction is “in”

- received from some node N

$\vdots \qquad\qquad \vdots$

Any message, including the first in the pattern, MAY trigger a fault message, which MUST have opposite direction.

This pattern has two paths. The first is identical to the previous pattern, with just a single message sent. The second includes a fault message returned in response to the original message, as shown below:



A specification that matches the above pattern is as follows:

$$
\begin{aligned}
SIMPLE\_SPEC\_ROO \mathrel{\widehat=} Notification \to (&Fault \to Stop \\
&\Box\ Stop)
\end{aligned}
$$

Expanding on $SIMPLE\_SPEC\_ROO$ to show the individual send and receive events gives this specification:

$$
\begin{aligned}
SPEC\_ROO \mathrel{\widehat=} sendNoti \to getNoti \to (&sendFault \to getFault \to Stop \\
&\Box\ Stop)
\end{aligned}
$$

$SPEC\_ROO$ can then be teased apart to give a description for the robust-out-only (ROO) and robust-in-only (RIO) ports:

$$
\begin{aligned}
&ROO \mathrel{\widehat=} sendNoti \to (ROO\_OK \\
&\qquad\qquad\qquad \Box\ getFault \to ROO\_FAULT) \\
&ROO\_OK \mathrel{\widehat=} Stop \\
&ROO\_FAULT \mathrel{\widehat=} Stop
\end{aligned}
$$

$$
\begin{aligned}
&RIO \mathrel{\widehat=} getNoti \to (RIO\_OK \\
&\qquad\qquad\qquad \sqcap\ sendFault \to RIO\_FAULT) \\
&RIO\_OK = Stop \\
&RIO\_FAULT = Stop
\end{aligned}
$$

These patterns also require a faux event to be inserted to give a detectable point at which the conversational thread leaves the port. Adding these gives the following templates for the port patterns.

$$
\begin{aligned}
&ROO \mathrel{\widehat=} sendNoti \rightarrow (ROO\_p1\\
&ROO\_p1 \mathrel{\widehat=} decThread \rightarrow ROO\_p2\\
&ROO\_p2 \mathrel{\widehat=} ROO\_p3 \,\Box\, ROO\_p4\\
&ROO\_p3 \mathrel{\widehat=} ROO\_OK\\
&ROO\_p4 \mathrel{\widehat=} getFault \rightarrow ROO\_FAULT\\
&ROO\_OK \mathrel{\widehat=} Stop\\
&ROO\_FAULT \mathrel{\widehat=} Stop
\end{aligned}
$$

$$
\begin{aligned}
&RIO \mathrel{\widehat=} getNoti \rightarrow (RIO\_p1\\
&RIO\_p1 \mathrel{\widehat=} decThread \rightarrow RIO\_p2\\
&RIO\_p2 \mathrel{\widehat=} RIO\_p3 \,\Box\, RIO\_p4\\
&RIO\_p3 \mathrel{\widehat=} RIO\_OK\\
&RIO\_p4 \mathrel{\widehat=} sendFault \rightarrow RIO\_FAULT\\
&RIO\_OK \mathrel{\widehat=} Stop\\
&RIO\_FAULT \mathrel{\widehat=} Stop
\end{aligned}
$$

Again, the templates are composed into a system to demonstrate that they behave identically to the specification.

$$
\begin{aligned}
&\alpha ROO = \{sendNoti, getFault\}\\
&\alpha RIO = \{getNoti, sendFault\}
\end{aligned}
$$

$$PORTS\_ROO \mathrel{\widehat=} ROO \;|||\; RIO$$

$$
\begin{aligned}
CONN\_ROO \mathrel{\widehat=}\; & sendNoti \rightarrow getNoti \rightarrow CONN\_ROO\\
& \Box\; sendFault \rightarrow getFault \rightarrow CONN\_ROO
\end{aligned}
$$

$$COMPOSED\_ROO \mathrel{\widehat=} PORTS\_ROO \,|[\, \alpha ROO, \alpha RIO \,]|\, CONN\_ROO$$

$$
\begin{aligned}
&COMPOSED\_ROO \setminus decThread \sqsubseteq \mathcal{M}_{UT} SPEC\_ROO\\
&SPEC\_ROO \;\sqsubseteq \mathcal{M}_{UT} COMPOSED\_ROO \setminus decThread
\end{aligned}
$$

I.1.2.2 Actual Templates and Useage

Below are the templates correctly formatted for use in the style along with notes regarding their usage.

```
1 roo
2 ROO = sendNoti -> ROO_p1
3 ROO_p1 = decThread -> ROO_p2
4 ROO_p2 = ROO_p3 [] ROO_p4
5 ROO_p3 = ROO_OK
6 ROO_p4 = getFault -> ROO_FAULT
7 ROO_OK = STOP
8 ROO_FAULT = STOP
```

1. Line 1 contains the template ID, this should not be altered;
2. All instances of “ROO”, except that on line 1, should be replaced with the “<componentID>-<portID>”.
3. “sendNoti” should be replaced with the ID of the message this port sends;
4. “getFault” should be replaced with the ID of the message this port might receive;
5. “STOP” on line 7 should be replaced with the ID where the process flows after this in the case where no fault message is received;
6. “STOP” on line 8 should be replaced with the ID where the process flows after this in the case where a fault message is received.

```
1 rio
2 RIO = getNoti -> RIO_p1
3 RIO_p1 = decThread -> RIO_p2
4 RIO_p2 = RIO_p3 [] RIO_p4
5 RIO_p3 = RIO_OK
6 RIO_p4 = sendFault -> RIO_FAULT
7 RIO_OK = STOP
8 RIO_FAULT = STOP
```

1. Line 1 contains the template ID, this should not be altered;
2. All instances of “RIO”, except that one line 1, should be replaced with the “<componentID>-<portID>”.
3. “getNoti” should be replaced with the ID of the message this port receives;
4. “sendFault” should be replaced with the ID of the fault message this port might send;
5. If the conversation thread is to leave this port before returning to complete the pattern, then the “RIO_p1” at the end of line 2 should be replaced with the ID of the process to be invoked;

6. "STOP" on line 7 should be replaced with the ID where the process flows if this port does not send a fault message.

7. "STOP" on line 8 should be replaced with the ID where the process flows after this port sends a fault message;

I.1.3 Solicit-Response - Request-Response

I.1.3.1 Template Derivation

The next patterns considered are the solicit-response / request-response patterns. The W3C description of solicit-response follows:

1. A message:
   * indicated by a Interface Message Reference component whose message label is "Out"and direction is "out"
   * sent to some node N
2. A message:
   * indicated by a Interface Message Reference component whose message label is "In" and direction is "in"
   * sent from node N

   ⋮ ⋮

Any message after the first in the pattern MAY be replaced with a fault message, which MUST have identical direction.

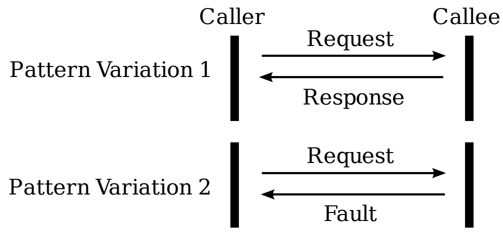
Here is the description of the matching request-response pattern:

1. A message:
   * indicated by a Interface Message Reference component whose message label is "In" and direction is "in"
   * received from some node N
2. A message:
   * indicated by a Interface Message Reference component whose message label is "Out" and direction is "out"
   * sent to node N

   ⋮ ⋮

Any message after the first in the pattern MAY be replaced with a fault message, which MUST have identical direction.

These patterns have two paths through them, a request message followed by a response or a request followed by a fault message. This is shown graphically below:



This is then converted into the following simple specification:

$$\begin{aligned} TRIV\_SPEC\_SOLI \ &\widehat{=}\ Request \rightarrow (Response \rightarrow Stop \\ &\qquad\qquad\qquad \Box\ Fault \rightarrow Stop) \end{aligned}$$

The simple specification is then expanded to show the individual message transmissions and receipts as follows:

$$\begin{aligned} SPEC\_SOLI \ &\widehat{=}\ sendReq \rightarrow getReq \rightarrow (sendRes \rightarrow getRes \rightarrow Stop \\ &\qquad\qquad\qquad\qquad\qquad \Box\ sendFault \rightarrow getFault \rightarrow Stop) \end{aligned}$$

Once again the specification is dissected and distributed into shorter lines resulting in templates for the solicit-response (SOLI) and request-response (REQR) patterns. These two patterns are the only ones that do not need any additional events adding to indicate when the conversation thread leaves the port and so no further modification are necessary:

$$\begin{aligned} SOLI \ &\widehat{=}\ sendReq \rightarrow SOLI\_P1 \\ SOLI\_P1 \ &\widehat{=}\ SOLI\_P2 \ \Box\ SOLI\_P3 \\ SOLI\_P2 \ &\widehat{=}\ getRes \rightarrow SOLI\_OK \\ SOLI\_P3 \ &\widehat{=}\ getFault \rightarrow SOLI\_FAULT \\ SOLI\_OK \ &\widehat{=}\ Stop \\ SOLI\_FAULT \ &\widehat{=}\ Stop \end{aligned}$$

$$\begin{aligned} REQR \ &\widehat{=}\ getReq \rightarrow REQR\_P1 \\ REQR\_P1 \ &\widehat{=}\ REQR\_P2 \ \Box\ REQR\_P3 \\ REQR\_P2 \ &\widehat{=}\ sendRes \rightarrow REQR\_OK \\ REQR\_P3 \ &\widehat{=}\ sendFault \rightarrow REQR\_FAULT \\ REQR\_OK \ &\widehat{=}\ Stop \\ REQR\_FAULT \ &\widehat{=}\ Stop \end{aligned}$$

The templates can be demonstrated to behave identically to the earlier specification by construction a system as follows and presenting it to the FDR model checker:

$$\begin{aligned}
\alpha \mathit{SOLI} &= \mathit{set sendReq}, \mathit{getRes}, \mathit{getFault} \\
\alpha \mathit{REQR} &= \mathit{set getReq}, \mathit{sendRes}, \mathit{sendFault}
\end{aligned}$$

$$\begin{aligned}
\mathit{CONN_SOLI} \mathrel{\widehat=}\ & \mathit{sendReq} \rightarrow \mathit{getReq} \rightarrow \mathit{CONN_SOLI} \\
& \Box\ \mathit{sendRes} \rightarrow \mathit{getRes} \rightarrow \mathit{CONN_SOLI} \\
& \Box\ \mathit{sendFault} \rightarrow \mathit{getFault} \rightarrow \mathit{CONN_SOLI}
\end{aligned}$$

$$\mathit{PORTS\_SOLI} \mathrel{\widehat=} \mathit{SOLI} \mathbin{|||} \mathit{REQR}$$

$$\mathit{COMPOSED\_SOLI} \mathrel{\widehat=} \mathit{PORTS\_SOLI} \mathbin{|[\, \alpha \mathit{SOLI}, \alpha \mathit{REQR}\,]|} \mathit{CONN\_SOLI}$$

$$\begin{aligned}
&\mathit{COMPOSED_SOLI} \sqsubseteq \mathcal{M}_{UT} \mathit{SPEC_SOLI} \\
&\mathit{SPEC_SOLI} \sqsubseteq \mathcal{M}_{UT} \mathit{COMPOSED_SOLI}
\end{aligned}$$

I.1.3.2 Actual Templates and Useage

```
1 soli
2 SOLI = sendReq -> SOLI_p1
3 SOLI_p1 = SOLI_p2 [] SOLI_p3
4 SOLI_p2 = getRes -> SOLI_OK
5 SOLI_p3 = getFault -> SOLI_FAULT
6 SOLI_OK = STOP
7 SOLI_FAULT = STOP
```

1. Line 1 contains the template ID, this should not be altered;
2. All instances of “SOLI”, except that one line 1, should be replaced with the “<componentID>-<portID>”.
3. “sendReq” should be replaced with the ID of the message this port sends;
4. “getRes” should be replaced with the ID of the normal response message this port receives;
5. “getFault” should be replaced with the ID of the fault message this port might receive;
6. If the process is to break out of this port, as described later, then the “SOLI_p1” at the end of line 2 should be replaced with the ID of the port to be moved to;
7. “STOP” on line 6 should be replaced with the ID where the process flows after this in the case where no fault message is received;
8. “STOP” on line 7 should be replaced with the ID where the process flows after this in the case where a fault message is received.

```
1 reqr
2 REQR = getReq -> REQR_p1
3 REQR_p1 = REQR_p2 [] REQR_p3
4 REQR_p2 = sendRes -> REQR_OK
5 REQR_p3 = sendFault -> REQR_FAULT
6 REQR_OK = STOP
7 REQR_FAULT = STOP
```

1. Line 1 contains the template ID, this should not be altered;
2. All instances of "REQR", except that one line 1, should be replaced with the "<componentID>-<portID>".
3. "getReq" should be replaced with the ID of the message this port receives;
4. "sendRes" should be replaced with the ID of the message this normally sends;
5. "sendFault" should be replaced with the ID of the fault message this port may send;
6. "STOP" on line 6 should be replaced with the ID where the process flows after this in the case where no fault message is sent;
7. "STOP" on line 7 should be replaced with the ID where the process flows after this in the case where a fault message is sent.

I.1.4 Out-optional-in - In-optional-out

The final pair of patterns are the out-optional-in / in-optional-out pairing. The description of out-optional-in follows:

1. A message:
   - indicated by a Interface Message Reference component whose message label is "Out" and direction is "out"
   - sent to some node N
2. An optional message:
   - indicated by a Interface Message Reference component whose message label is "In" and direction is "in"
   - sent from node N

   ⋮ ⋮

   Any message, including the first in the pattern, MAY trigger a fault message, which MUST have opposite direction.

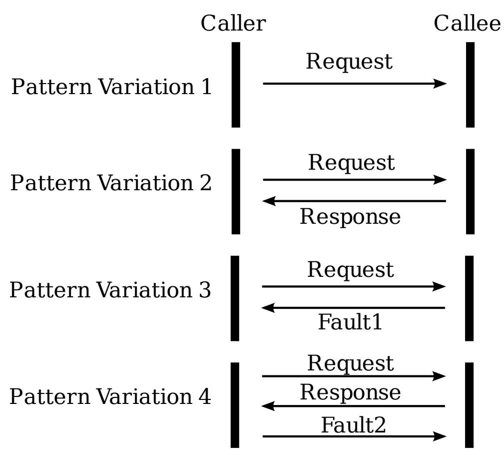
The description of the in-optional-out pattern follows:

1. A message:
   - indicated by a Interface Message Reference component whose message label is "In" and direction is "in"
   - received from some node N
2. An optional message:
   - indicated by a Interface Message Reference component whose message label is "Out" and direction is "out"
   - sent to node N

$\vdots \qquad\qquad \vdots$

Any message, including the first in the pattern, MAY trigger a fault message, which MUST have opposite direction.

This pattern has four paths through it. The first is a single message with no response. The second path is a message with a normal response while the third is a message with a fault message returned. The final path extends the second path, allowing the port receiving the response message to respond with a fault message if required. These patterns are shown below.



This is represented by the following specification:

$$
\begin{aligned}
\mathit{SIMPLE\_SPEC\_OOI} \mathrel{\widehat{=}} \mathit{Request} \rightarrow (&\mathit{Stop}\\
&\Box\ \mathit{Fault1} \rightarrow \mathit{Stop}\\
&\Box\ \mathit{Response} \rightarrow (\mathit{Fault2} \rightarrow \mathit{Stop}\\
&\qquad\qquad\qquad \Box\ \mathit{Stop}))
\end{aligned}
$$

The simple specification can then be expanded to represent all message send and receive events as follows:

$$
\begin{aligned}
SPEC\_OOI \mathrel{\widehat{=}} sendReq \rightarrow getReq \rightarrow (&Stop \\
&\Box\ sendFault1 \rightarrow getFault1 \rightarrow Stop \\
&\Box\ sendRes \rightarrow getRes \rightarrow (sendFault2 \rightarrow getFault2 \rightarrow Stop \\
&\qquad\qquad\qquad\qquad\qquad \Box\ Stop))
\end{aligned}
$$

After separating out the messages each port sends and receives the following templates for the out-optional-in (OOI) and in-optional-out (IOO) patterns can be found:

$$
\begin{aligned}
&OOI \mathrel{\widehat{=}} sendReq \rightarrow OOI\_p1 \\
&OOI\_p1 \mathrel{\widehat{=}} OOI\_p2 \ \Box\ OOI\_p3 \ \Box\ OOI\_p4 \\
&OOI\_p2 \mathrel{\widehat{=}} getFault \rightarrow OOI\_FAULT \\
&OOI\_p3 \mathrel{\widehat{=}} getRes \rightarrow OOI\_p5 \\
&OOI\_p4 \mathrel{\widehat{=}} OOI\_NORES \\
&OOI\_p5 \mathrel{\widehat{=}} OOI\_p6 \ \Box\ OOI\_p7 \\
&OOI\_p6 \mathrel{\widehat{=}} OOI\_RES \\
&OOI\_p7 \mathrel{\widehat{=}} sendFault2 \rightarrow OOI\_RESFAULT \\
&OOI\_FAULT \mathrel{\widehat{=}} Stop \\
&OOI\_RES \mathrel{\widehat{=}} Stop \\
&OOI\_RESFAULT \mathrel{\widehat{=}} Stop \\
&OOI\_OK \mathrel{\widehat{=}} Stop
\end{aligned}
$$

$$
\begin{aligned}
&IOO \mathrel{\widehat{=}} getReq \rightarrow IOO\_p1 \\
&IOO\_p1 \mathrel{\widehat{=}} IOO\_p2 \ \Box\ IOO\_p3 \ \Box\ IOO\_p4 \\
&IOO\_p2 \mathrel{\widehat{=}} IOO\_OK \\
&IOO\_p3 \mathrel{\widehat{=}} sendFault \rightarrow IOO\_FAULT \\
&IOO\_p4 \mathrel{\widehat{=}} sendRes \rightarrow IOO\_p5 \\
&IOO\_p5 \mathrel{\widehat{=}} IOO\_p6 \ \Box\ IOO\_p7 \\
&IOO\_p6 \mathrel{\widehat{=}} getFault2 \rightarrow IOO\_RESFAULT \\
&IOO\_p7 \mathrel{\widehat{=}} IOO\_RES \\
&IOO\_OK \mathrel{\widehat{=}} Stop \\
&IOO\_FAULT \mathrel{\widehat{=}} Stop \\
&IOO\_RESFAULT \mathrel{\widehat{=}} Stop \\
&IOO\_RESOK \mathrel{\widehat{=}} Stop
\end{aligned}
$$

The out-optional-in and in-optional-out patterns both require a faux event to be included to act as the point at which the conversational thread leaves the port. Adding this results in the following two templates:

$OOI \mathrel{\widehat=} sendReq \rightarrow OOI\_p1$

$OOI\_p1 \mathrel{\widehat=} decThread \rightarrow OOI\_p2$

$OOI\_p2 \mathrel{\widehat=} OOI\_p3 \,\Box\, OOI\_p4 \,\Box\, OOI\_p5$

$OOI\_p3 \mathrel{\widehat=} getFault \rightarrow OOI\_FAULT$

$OOI\_p4 \mathrel{\widehat=} getRes \rightarrow OOI\_p6$

$OOI\_p5 \mathrel{\widehat=} OOI\_NORES$

$OOI\_p6 \mathrel{\widehat=} OOI\_p7 \,\Box\, OOI\_p8$

$OOI\_p7 \mathrel{\widehat=} OOI\_RES\;\; OOI\_p8 \mathrel{\widehat=} sendFault2 \rightarrow OOI\_RESFAULT$

$OOI\_FAULT \mathrel{\widehat=} Stop$

$OOI\_RES \mathrel{\widehat=} Stop$

$OOI\_RESFAULT \mathrel{\widehat=} Stop$

$OOI\_OK \mathrel{\widehat=} Stop$

$IOO \mathrel{\widehat=} getReq \rightarrow IOO\_p1$

$IOO\_p1 \mathrel{\widehat=} decThread$

IOO_p2

$IOO\_p2 \mathrel{\widehat=} IOO\_p3 \,\Box\, IOO\_p4 \,\Box\, IOO\_p5$

$IOO\_p3 \mathrel{\widehat=} IOO\_OK$

$IOO\_p4 \mathrel{\widehat=} sendFault \rightarrow IOO\_FAULT$

$IOO\_p5 \mathrel{\widehat=} sendRes \rightarrow IOO\_p6$

$IOO\_p6 \mathrel{\widehat=} IOO\_p7 \,\Box\, IOO\_p8$

$IOO\_p7 \mathrel{\widehat=} getFault2 \rightarrow IOO\_RESFAULT$

$IOO\_p8 \mathrel{\widehat=} IOO\_RES$

$IOO\_OK \mathrel{\widehat=} Stop$

$IOO\_FAULT \mathrel{\widehat=} Stop$

$IOO\_RESFAULT \mathrel{\widehat=} Stop$

$IOO\_RESOK \mathrel{\widehat=} Stop$

Once again, the templates can be demonstrated to be correct to the specification by constructing the following system and evaluating the assertions:

$$\alpha OOI = \{sendReq, getRes, getFault, sendFault2\}$$
$$\alpha IOO = \{getReq, sendRes, sendFault1, getFault2\}$$

$$PORTS\_OOI \mathrel{\widehat{=}} OOI \mathbin{|||} IOO$$

$$\begin{aligned} CONN\_OOI \mathrel{\widehat{=}}\ & sendReq \rightarrow getReq \rightarrow CONN\_OOI \\ & \Box\ sendRes \rightarrow getRes \rightarrow CONN\_OOI \\ & \Box\ sendFault \rightarrow getFault \rightarrow CONN\_OOI \\ & \Box\ sendFault2 \rightarrow getFault2 \rightarrow CONN\_OOI \end{aligned}$$

$$COMPOSED\_OOI \mathrel{\widehat{=}} PORTS\_OOI \mathbin{|[\, \alpha OOI, \alpha IOO \,]|} CONN\_OOI$$

$$COMPOSED\_OOI \setminus decThread \sqsubseteq \mathcal{M}_{UT} SPEC\_OOI$$
$$SPEC\_OOI \sqsubseteq \mathcal{M}_{UT} COMPOSED\_OOI \setminus decThread$$

I.1.4.1 Actual Templates and Useage

```
1  ooi
2  OOI = sendReq -> OOI_p1
3  OOI_p1 = decThread -> OOI_p2
4  OOI_p2 = OOI_p3 [] OOI_p4 [] OOI_p5
5  OOI_p3 = getFault -> OOI_FAULT
6  OOI_p4 = getRes -> OOI_p6
7  OOI_p5 = OOI_NORES
8  OOI_p6 = OOI_p7 [] OOI_p8
9  OOI_p7 = OOI_RES
10 OOI_p8 = sendFault2 -> OOI_RESFAULT
11 OOI_FAULT = STOP
12 OOI_NORES = STOP
13 OOI_RES = STOP
14 OOI_RESFAULT = STOP
```

1. Line 1 contains the template ID, this should not be altered;
2. All instances of "OOI", except the one line 1, should be replaced with the "<componentID>-<portID>".
3. "sendReq" should be replaced with the ID of the message this port sends;
4. "getRes" should be replaced with the ID of the normal response message this port receives;
5. "getFault" should be replaced with the ID of the fault message this port might receive;
6. "sendFault2" should be replaced with the ID of the fault message this port might send;
7. If the conversation is to break out of this port, then the "OOI_p2" on line 3 should be replaced with the name of the process to move to. It is this point and not "OOI_p1" as this ensure that the faux event *decThread* is executed and thus keeps the thread counting correct;

8. “STOP” on line 11 should be replaced with the ID where the process flows after this in the case this port receives a fault message;

9. “STOP” on line 12 should be replaced with the ID where the process flows after this in the case where this port does not receive any response;

10. “STOP” on line 13 should be replaced with the ID where the process flows after this in the case where this port receives a normal response message;

11. “STOP” on line 14 should be replaced with the ID where the process flows after this in the case where this port has to send a fault message.

```
1  ioo
2  IOO = getReq -> IOO_p1
3  IOO_p1 = decThread -> IOO_p2
4  IOO_p2 = IOO_p3 [] IOO_p4 [] IOO_p5
5  IOO_p3 = IOO_OK
6  IOO_p4 = sendFault -> IOO_FAULT
7  IOO_p5 = sendRes -> IOO_p6
8  IOO_p6 = IOO_p7 [] IOO_p8
9  IOO_p7 = getFault2 -> IOO_RESFAULT
10 IOO_p8 = IOO_RES
11 IOO_OK = STOP
12 IOO_FAULT = STOP
13 IOO_RESFAULT = STOP
14 IOO_RES = STOP
```

1. Line 1 contains the template ID, this should not be altered;

2. All instances of “IOO”, except the one line 1, should be replaced with the “<componentID>-<portID>”.

3. “getReq” should be replaced with the ID of the message this port receives;

4. “sendRes” should be replaced with the ID of the normal response message this port sends;

5. “sendFault” should be replaced with the ID of the fault message this port might send;

6. “getFault2” should be replaced with the ID of the fault message this port might receive;

7. “STOP” on line 11 should be replaced with the ID where the process flows after this in the case this port receives a request but does not send any response ;

8. “STOP” on line 12 should be replaced with the ID where the process flows after this in the case where this sends a fault message in response to the request;

9. “STOP” on line 13 should be replaced with the ID where the process flows after this in the case where this port sends a normal response but then receives a fault message;

10. "STOP" on line 14 should be replaced with the ID where the process flows after this in the case where this port sends a normal response and does not receive a resulting fault message.

I.2 Port Template Linking

One of the motivations behind the port CSP templates was to make explicit how and where they are to be modified to represent how the conversational thread might flow between ports on a component. Two types of flow are implemented by altering the names in the port CSP templates, these are "sequential flow" and "breaking out".

I.2.1 Sequential Flow

A sequential flow is where, after passing through the message exchange pattern of one port the choreography moves onto another port and there is no choice about the identity of that port.

This will be illustrated with a simple client-server example. In this example both the client and server has ports labelled A and B and both components expect to interact on port A and then on port B. A UML sequence diagram indicating this behaviour can be seen in Figure I.1.

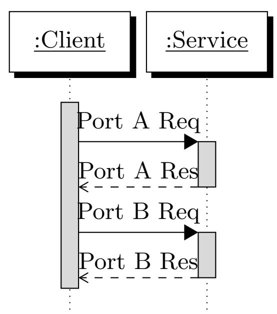


Figure I.1: A sequential flow where both components expect to interact on port A and then port B

Only the client CSP will be shown as the server has a similar structure.

The central CSP of the client causes port A to be active initially:

$$CLIENT \mathrel{\widehat{=}} CLIENT\_THREAD$$
$$CLIENT\_THREAD \mathrel{\widehat{=}} PORT\_A$$

$Port\_A$ on the client has been modified so that the two end points point to the next port in the sequence, in this case $PORT\_B$:

$PORT\_A \mathrel{\widehat=} getReq \rightarrow PORT\_A\_p1$

$PORT\_A\_p1 \mathrel{\widehat=} PORT\_A\_p2 \mathrel{\Box} PORT\_A\_p3$

$PORT\_A\_p2 \mathrel{\widehat=} sendRes \rightarrow PORT\_A\_OK$

$PORT\_A\_p3 \mathrel{\widehat=} sendFault \rightarrow PORT\_A\_FAULT$

$PORT\_A\_OK \mathrel{\widehat=} PORT\_B$

$PORT\_A\_FAULT \mathrel{\widehat=} PORT\_B$

Port_B is the end of the conversation and returns the client back to its starting point, which is this case is the *CLIENT_THREAD* process:

$PORT\_B \mathrel{\widehat=} sendReq \rightarrow PORT\_B\_p1$

$PORT\_B\_p1 \mathrel{\widehat=} PORT\_B\_p2 \mathrel{\Box} PORT\_B\_p3$

$PORT\_B\_p2 \mathrel{\widehat=} getRes \rightarrow PORT\_B\_OK$

$PORT\_B\_p3 \mathrel{\widehat=} getFault \rightarrow PORT\_B\_FAULT$

$PORT\_B\_OK \mathrel{\widehat=} CLIENT\_THREAD$

$PORT\_B\_FAULT \mathrel{\widehat=} CLIENT\_THREAD$

Such a component will loop through the two ports indefinitely.

I.2.2 Breaking Out

The second type of flow implemented purely within the port CSP is breaking out. This is where, after receiving a message a port directs the conversation toward another port and awaits a response before completing its interaction. An example of such a situation would be a broker or mediator as described in the car parking example in Chapter 6. An illustration of such a breakout is shown in Figure I.2.

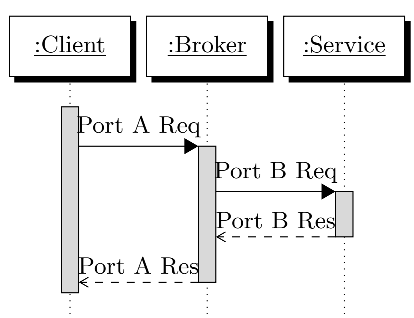


Figure I.2: A break out flow where a message received by a port triggers the invocation of another port on the same component to obtain a result.

In this example the CSP included in the BROKER component is highlighted. Its central CSP makes port A initially active:

$$BROKER \mathrel{\widehat{=}} BROKER\_THREAD$$
$$BROKER\_THREAD \mathrel{\widehat{=}} PORT\_A$$

In this case port A receives a message and then has to invoke port B, this is achieved by altering the name at the end of the first line in the actual CSP. Normally in the template this line would direct the process to *PORT_A_p1* however now it breaks out of this pattern and move of *PORT_B* instead.

$$PORT\_A \mathrel{\widehat{=}} getReq \rightarrow PORT\_B$$
$$PORT\_A\_p1 \mathrel{\widehat{=}} PORT\_A\_p2 \Box PORT\_A\_p3$$
$$PORT\_A\_p2 \mathrel{\widehat{=}} sendRes \rightarrow PORT\_A\_OK$$
$$PORT\_A\_p3 \mathrel{\widehat{=}} sendFault \rightarrow PORT\_A\_FAULT$$
$$PORT\_A\_OK \mathrel{\widehat{=}} BROKER\_THREAD$$
$$PORT\_A\_FAULT \mathrel{\widehat{=}} BROKER\_THREAD$$

No change is needed to the CSP of port B to receive the thread from port A, however the two outcomes at the end of the template are altered. Instead of pointing to the next port in the sequence as in the previous example now they point so they invoke the relevant message in port A. specifically the port B outcome *PORT_B_OK* points to *PORT_A_p2* which sends a normal response to the client while the *PORT_B_FAULT* outcome points to *PORT_A_p3* to send a fault back to the client.

$$PORT\_B \mathrel{\widehat{=}} sendReq \rightarrow PORT\_B\_p1$$
$$PORT\_B\_p1 \mathrel{\widehat{=}} PORT\_B\_p2 \Box PORT\_B\_p3$$
$$PORT\_B\_p2 \mathrel{\widehat{=}} getRes \rightarrow PORT\_B\_OK$$
$$PORT\_B\_p3 \mathrel{\widehat{=}} getFault \rightarrow PORT\_B\_FAULT$$
$$PORT\_B\_OK \mathrel{\widehat{=}} PORT\_A\_p2$$
$$PORT\_B\_FAULT \mathrel{\widehat{=}} PORT\_A\_p3$$

I.3 Central CSP Templates

While much of the detail concerning the interactions of a component are described in the port CSP, the central CSP dictates more coarse grained properties such as how many threads of control a component possesses and what those threads are initially willing to do.

I.3.1 Single Thread

The simplest of the central CSP templates applies when a component has only a single thread of control and that thread is initially only willing to interact on a single port.

The client component described earlier in the sequential flow example, shown in Figure I.1 uses this template. In that example the client component was initially only willing the interact on port A, its central CSP is as follows:

$$CLIENT \mathrel{\widehat{=}} CLIENT\_THREAD$$
$$CLIENT\_THREAD \mathrel{\widehat{=}} PORT\_A$$

The first line in this description starts by defining a process with the same ID as the component in which it exists. This process is defined as behaving as *CLIENT_THREAD*. This named process is then defined on the second line as behaving as *PORT_A*.

In effect this defines a process called *CLIENT* that behaves as *PORT_A* and so contains what looks initially like redundant definitions, however this structure is important when multiple threads are considered, as will be described later.

In terms of altering this template to fit a component:

1. All instances of the string *CLIENT* should be replaced with the ID of the component in which this central CSP exists;

2. *PORT_A* should be replaced with the ID of the first port the component wished to interact upon.

I.3.2 Single Thread With Choice of Ports

The first extension of the previous case is a component with a single thread of control but the thread is willing to interact on one of two or more ports initially.

An example of such a component could be a service that provides two distinct functions A and B but the functions are mutually exclusive. In this case a client may choose to interact with A or B as shown in Figure I.3.

Figure I.3: Sequence diagrams representing the choices of port offered by the service component.

In this case the service component would contain the following central CSP:

$$SERVICE \mathrel{\widehat{=}} SERVICE\_THREAD$$
$$SERVICE\_THREAD \mathrel{\widehat{=}} PORT\_A \mathbin{\Box} PORT\_B$$

The structure here is identical to the previous example, the only difference being that *SERVICE_THREAD* is defined as having a choice of behaving as *PORT_A* or as *PORT_B*. If further port choices were available to the thread then these can be appended to the list separated by the external choice operator as below:

$$SERVICE \mathrel{\widehat{=}} SERVICE\_THREAD$$
$$SERVICE\_THREAD \mathrel{\widehat{=}} PORT\_A \mathbin{\Box} PORT\_B \mathbin{\Box} PORT\_C$$

In terms of altering this template to fit a component:

1. All instances of the string *SERVICE* should be replaced with the ID of the component in which this central CSP exists;
2. *PORT_A*,*PORT_B* and *PORT_C* should be replaced with the IDs of the initially active ports, adding as many IDs as required.

I.3.3 Multiple Identical Threads

The other extension of the initial central CSP is to consider a component that has multiple identical threads of control, where identical refers to the choreography the thread expects.

Returning once again to the initial sequential flow example where the client component contained a single thread that was initially willing to interact on port A. A version of this component that contains two threads of control can be defined by using the following central CSP:

$$CLIENT \mathrel{\widehat{=}} CLIENT\_THREAD \mathbin{|||} CLIENT\_THREAD$$
$$CLIENT\_THREAD \mathrel{\widehat{=}} PORT\_A$$

The additional thread is created by adding an additional reference to the *CLIENT_THREAD* process to the component description line, the references are separated by the interleave operator to indicate that they do not synchronise on any events.

The modifications to specialise this template for a particular component are identical to those listed in the sequential flow section with one addition. To add additional identical threads to the component, add the required number of references to *CLIENT_THREAD* separated by interleave operators as shown below:

$$CLIENT \mathrel{\widehat{=}} CLIENT\_THREAD \mathbin{|||} CLIENT\_THREAD \mathbin{|||} CLIENT\_THREAD$$
$$CLIENT\_THREAD \mathrel{\widehat{=}} PORT\_A$$

At this point it is possible to see why the central CSP includes the seemingly redundant separation of between defining a process with the component name that simply invokes one or more instances of a thread process. If in the above example the outcomes of port A directed the process flow back up to *CLIENT* then the process would move back to its initial state of being ready to interact on port A, but at the same time two new process threads would be created that would also be ready to interact on port A. These duplicate threads would in fact be created each time and invocation of port A completes. This creates a situation where the process will attempt to create an infinite number of threads making model checking impossible. However if the outcomes of port A direct the process to the *CLIENT_THREAD* process then this has the effect to returning the conversation to its original point but without the side effect of creating extra threads and thus allowing the model checking to complete.

I.3.4 Multiple Diverse Threads

In the single thread with multiple choice example earlier it was assumed that the two functions A and B were mutually exclusive and so only a single thread was provided. If this is not the case then it might be desirable to offer both functions simultaneously. Such behaviour is defined in the central CSP as follows:

$$SERVICE \mathrel{\widehat{=}} SERVICE\_THREAD\_A \mathbin{|||} SERVICE\_THREAD\_B$$
$$SERVICE\_THREAD\_A \mathrel{\widehat{=}} PORT\_A$$
$$SERVICE\_THREAD\_B \mathrel{\widehat{=}} PORT\_B$$

The key points in this template are firstly that a definition is needed for the start point of each thread of control. In this case the processes *SERVICE_THREAD_A* and *SERVICE_THREAD_B* perform this function. The second key point is that the processes are then referenced on the first line and are separated by interleave operators. There should be one reference for each instance of a thread of each type that will exist in the component, for example the following would define a service component with one thread A and two thread Bs:

$$SERVICE \mathrel{\widehat{=}} SERVICE\_THREAD\_A \mathbin{|||} SERVICE\_THREAD\_B \mathbin{|||} SERVICE\_THREAD\_B$$
$$SERVICE\_THREAD\_A \mathrel{\widehat{=}} PORT\_A$$
$$SERVICE\_THREAD\_B \mathrel{\widehat{=}} PORT\_B$$

It should be noted that neither the order in which the thread instances are referenced on the first line nor the order in which they are defined on the followings are significant.

I.3.5 Branching

A branching flow is one where after executing the message exchanges associated with one port the conversation may then interact one of two or more ports. A simple example would be a service that after interacting on port A, which could be a login, allows the use of functions B or C. In this case the method shown above for describing a process that starts with the option of choosing one of two ports cannot help as the process has already moved beyond that point. Instead the suggestion is to make an alteration to both the port A CSP and add an entry to the central CSP.

The CSP in port A would be as follows:

$$PORT\_A \widehat{=} getReq \rightarrow PORT\_A\_p1$$
$$PORT\_A\_p1 \widehat{=} PORT\_A\_p2 \ \Box \ PORT\_A\_p3$$
$$PORT\_A\_p2 \widehat{=} sendRes \rightarrow PORT\_A\_OK$$
$$PORT\_A\_p3 \widehat{=} sendFault \rightarrow PORT\_A\_FAULT$$
$$PORT\_A\_OK \widehat{=} SERVICE\_BRANCH$$
$$PORT\_A\_FAULT \widehat{=} SERVICE\_THREAD$$

The two outcome lines, $PORT\_A\_OK$ and $PORT\_A\_FAULT$ are pointed to a new process called $SERVICE\_BRANCH$ rather than to a specific port. The central CSP for this port would then be:

$$SERVICE \widehat{=} SERVICE\_THREAD$$
$$SERVICE\_THREAD \widehat{=} PORT\_A$$
$$SERVICE\_BRANCH \widehat{=} PORT\_B \ \Box \ PORT\_C$$

So the actual branching of the process flow takes places in the $SERVICE\_BRANCH$ process in the central CSP rather than in the port CSP.

I.3.6 Looping

Looping can be represented in the style in a very similar way to branching. An example of its use would be a catalogue service that requires a login on port A, then allows zero or more uses of a function on port B before finally expecting a client to logout. This behaviour is represented in Figure I.4.

The central CSP of the client component would consist of a single thread that starts by wanting to interact on the login port. The central CSP also includes a $LOGGED\_IN$ process that is part of

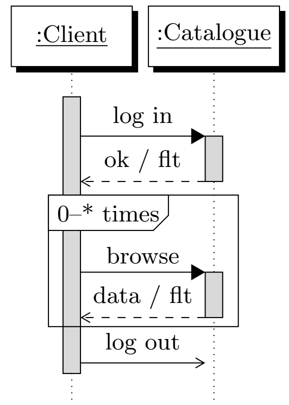


Figure I.4: Sequence diagram showing how a service might expect a conversation to flow if it included looping.

the looping construct, this will be described later.

$$CLIENT \mathrel{\widehat{=}} CLIENT\_THREAD$$
$$CLIENT\_THREAD \mathrel{\widehat{=}} LOG\_IN$$
$$LOGGED\_IN \mathrel{\widehat{=}} BROWSE \mathbin{\Box} LOG\_OUT$$

The first active port on the client, LOG_IN, gives and example of how the process flow can be differently directed based upon the type of message returned. In this case if a normal response message is received then the process moves onto a process named $LOGGED\_IN$, while if a fault message is returned the client returns to the initial point in the choreography as defined in the process $CLIENT\_THREAD$.

$$LOG\_IN \mathrel{\widehat{=}} login \rightarrow LOG\_IN\_p1$$
$$LOG\_IN\_p1 \mathrel{\widehat{=}} LOG\_IN\_p2 \mathbin{\Box} LOG\_IN\_p3$$
$$LOG\_IN\_p2 \mathrel{\widehat{=}} ok \rightarrow LOG\_IN\_OK$$
$$LOG\_IN\_p3 \mathrel{\widehat{=}} flt \rightarrow LOG\_IN\_FAULT$$
$$LOG\_IN\_OK \mathrel{\widehat{=}} LOGGED\_IN$$
$$LOG\_IN\_FAULT \mathrel{\widehat{=}} CLIENT\_THREAD$$

The $LOGGED\_IN$ process allows a choice of two processes, $BROWSE$ or LOG_OUT. Assuming the browse option is taken then the process will follow the that port's CSP:

$BROWSE \mathrel{\widehat=} browse \rightarrow BROWSE\_p1$

$BROWSE\_p1 \mathrel{\widehat=} BROWSE\_p2 \mathbin{\Box} BROWSE\_p3$

$BROWSE\_p2 \mathrel{\widehat=} data \rightarrow BROWSE\_OK$

$BROWSE\_p3 \mathrel{\widehat=} flt \rightarrow BROWSE\_FAULT$

$BROWSE\_OK \mathrel{\widehat=} LOGGED\_IN$

$BROWSE\_FAULT \mathrel{\widehat=} LOGGED\_IN$

The key point of this CSP is that both outcomes direct the conversation to follow the $LOGGED\_IN$ process once again. This again allows a choice of whether to browse or logout. The process may follow the browse option zero or more times before following the logout option. The logout port uses the notification pattern and simply sends a single message to the catalogue before performing the faux *decThread* discussed earlier and then returning the process to the initial state as defined in $CLIENT\_THREAD$.

$LOG\_OUT \mathrel{\widehat=} logout \rightarrow LOG\_OUT\_p1$

$LOG\_OUT\_p1 \mathrel{\widehat=} decThread \rightarrow LOG\_OUT\_OK$

$LOG\_OUT\_OK \mathrel{\widehat=} CLIENT\_THREAD$

This concludes a description of all the templates and constructs required to utilise the style.