

Software Development in the *Post-PC* Era: Towards Software Development as a Service

Sami Alajrami

*Submitted for the degree of Doctor of
Philosophy in the School of Computing
Science, Newcastle University*

May 2017

In memory of my loving parents

DECLARATION

I declare that this thesis is my own work unless otherwise stated. No part of this thesis has previously been submitted for a degree or any other qualification at Newcastle University or any other institution.

Sami Alajrami

May 2017

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and support of several people. I owe the successful completion of this thesis to those individuals. First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Alexander Romanovsky who believed in this work when I had moments of doubt and whose patience, commitment, advice and continuous encouragement have helped me to become an independent researcher. I am also very grateful to the school of computing science at Newcastle University and the head of school Prof. Aad van Moorsel for giving me the opportunity to undertake this work. I would like also to thank my co-supervisor Dr. Barbara Gallina whose expertise and generous guidance have helped shaping this work.

It is a pleasure to thank all those who contributed to this thesis through fruitful discussions, suggestions and kind advice. Especially, Prof. Paul Watson, Dr. Jacek Cala, Dr. Simon Woodman, Prof. Andreas Roth, Dr. Victor Khomenko, Dr. Leo Freitas, Dr. Juan Perna and Dr. Rajiv Ranjan. I would also like to thank Irfan Slijvo and Petter Isberg for the fruitful collaboration.

This acknowledgement cannot be complete without expressing my deepest gratitude to all friends and family. Specially; my best friend and true sister, Maryam, your selfless time and care were sometimes all that kept me going. I am forever grateful.

Ehsan, thanks for being a true brother and for the good times and the nice cakes.

Maher and Mohammad, you made me feel welcomed. Your advice and experience have always been a great resource. Thank you.

David, Razgar, Diego, Rani, Ayman, Khalid and Adham, I am grateful to have such good friends.

My siblings, you have always been a source of motivation. Thank you.

Last, but not least, special thanks to my loving parents. Mom, your selfless encouragement is what made me take this journey in the first place. You left this world but never left my thoughts. Dad, your memory has always been with me. You are both missed.

ABSTRACT

Engineering software systems is a complex task which involves various stakeholders and requires planning and management to succeed. As the role of software in our daily life is increasing, the complexity of software systems is increasing. Throughout the short history of software engineering as a discipline, the development practises and methods have rapidly evolved to seize opportunities enabled by new technologies (e.g., the Internet) and to overcome economical challenges (e.g., the need for cheaper and faster development).

Today, we are witnessing the *Post-PC* era. An era which is characterised by mobility and services. An era which removes organisational and geographical boundaries. An era which changes the functionality of software systems and requires alternative methods for conceiving them.

In this thesis, we envision to execute software development processes in the cloud. Software processes have a software production aspect and a management aspect. To the best of our knowledge, there are no academic nor industrial solutions supporting the entire software development process life-cycle (from both production and management aspects and its tool-chain execution in the cloud).

Our vision is to use the cloud economies of scale and leverage Model-Driven Engineering (MDE) to integrate production and management aspects into the development process. Since software processes are seen as workflows, we investigate using existing Workflow Management Systems to execute software processes and we find that these systems are not suitable. Therefore, we propose a reference architecture for *Software Development as a Service (SDaaS)*. **The SDaaS reference architecture is the first proposal which fully supports development of complex software systems in the cloud.**

In addition to the reference architecture, we investigate three specific related challenges and propose novel solutions addressing them. These challenges are:

- **Modelling & enacting cloud-based executable software processes.** Executing software processes in the cloud can bring several benefits to software develop-

ment. In this thesis, we discuss the benefits and considerations of cloud-based software processes and introduce a modelling language for modelling such processes. We refer to this language as *EXE-SPEM*. It extends the Software and Systems Process Engineering (SPEM2.0) OMG standard to support creating cloud-based executable software process models. Since EXE-SPEM is a visual modelling language, we introduce an XML notation to represent EXE-SPEM models in a machine-readable format and provide mapping rules from EXE-SPEM to this notation. We demonstrate this approach by modelling an example software process using EXE-SPEM and mapping it to the XML notation. Software process models expressed in this XML format can then be enacted in the proposed SDaaS architecture.

- **Cost-efficient scheduling of software processes execution in the cloud.** Software process models are enacted in the SDaaS architecture as workflows. We refer to them sometimes as *Software Workflows*. Once we have executable software process models, we need to schedule them for execution. In a setting where multiple software workflows (and their activities) compete for shared computational resources (workflow engines), scheduling workflow execution becomes important. Workflow scheduling is an NP-hard problem which refers to the allocation of sufficient resources (human or computational) to workflow activities. The schedule impacts the workflow makespan (execution time) and cost as well as the computational resources utilisation. The target of the scheduling is to reduce the process execution cost in the cloud without significantly affecting the process makespan while satisfying the special requirements of each process activity (e.g., executing on a private cloud). We adapt three workflow scheduling algorithms to fit for SDaaS and propose a fourth one; the *Proportional Adaptive Task Schedule*. The algorithms are then evaluated through simulation. The simulation results show that the our proposed algorithm saves between 19.74% and 45.78% of the execution cost, provides best resource (VM) utilisation and provides the second best makespan compared to the other presented algorithms.
- **Evaluating the SDaaS architecture using a case study from the safety-critical systems domain.** To evaluate the proposed SDaaS reference architecture, we instantiate a proof-of-concept implementation of the architecture. This imple-

mentation is then used to enact safety-critical processes as a case study.

Engineering safety-critical systems is a complex task which involves multiple stakeholders. It requires shared and scalable computation to systematically involve geographically distributed teams. In this case study, we use EXE-SPEM to model a portion of a process (namely; the Preliminary System Safety Assessment - PSSA) adapted from the ARP4761 [2] aerospace standard. Then, we enact this process model in the proof-of-concept SDaaS implementation.

By using the SDaaS architecture, we demonstrate the feasibility of our approach and its applicability to different domains and to customised processes. We also demonstrate the capability of EXE-SPEM to model cloud-based executable processes. Furthermore, we demonstrate the added value of the process models and the process execution provenance data recorded by the SDaaS architecture. This data is used to automate the generation of safety cases argument fragments. Thus, reducing the development cost and time. Finally, the case study shows that we can integrate some existing tools and create new ones as activities used in process models.

The proposed SDaaS reference architecture (combined with its modelling, scheduling and enactment capabilities) brings the benefits of the cloud to software development. It can potentially save software production cost and provide an accessible platform that supports collaborating teams (potentially across different locations). The executable process models support unified interpretation and execution of processes across team(s) members. In addition, the use of models provide managers with global awareness and can be utilised for quality assurance and process metrics analysis and improvement.

We see the contributions provided in this thesis as a first step towards an alternative development method that uses the benefits of cloud and Model-Driven Engineering to overcome existing challenges and open new opportunities. However, there are several challenges that are outside the scope of this study which need to be addressed to allow full support of the SDaaS vision (e.g., supporting interactive workflows). The solutions provided in this thesis address only part of a bigger vision. There is also a need for empirical and usability studies to study the impact of the SDaaS architecture on both the produced products (in terms of quality, cost, time, etc.) and the participating stakeholders.

PUBLICATIONS

Portions of the work presented within this thesis have been documented in the following publications:

CONFERENCE PAPERS

1. **S Alajrami**, B Gallina, A Romanovsky, *EXE-SPEM: Towards Cloud-based Executable Software Process Models*. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELWARD'16), Rome, Italy, February 19-21, pages 517-526. Scitepress, 2016.
2. **S Alajrami**, B Gallina, I Sljivo, A Romanovsky, P Isberg: *Towards Cloud-based Enactment of Safety-related Processes*. In A Skavhaug, J Guiochet, and F Bitsch, editors, Computer Safety, Reliability, and Security: 35th International Conference, (SAFECOMP'16), Trondheim, Norway, September 21-23, Proceedings, pages 309-321. Springer, 2016.
3. **S Alajrami**, A Romanovsky, B Gallina: *Software Development in the Post-PC Era: Towards Software Development as a Service*. In P Abrahamsson and A Jedlitschka, editors, the 17th International Conference on Product-Focused Software Process Improvement, (PROFES'16), Trondheim, Norway, November 22-24, Proceedings. Springer, 2016.
4. **S Alajrami**, B Gallina, A Romanovsky: *Cost-Aware Scheduling of Software Processes Execution in the Cloud*. In: the 43rd Euromicro Conference on Software Engineering and Advanced Applications. (Submitted).

WORKSHOP PAPERS

1. **S Alajrami**, A Romanovsky, P Watson, and A Roth. *Towards Cloud-based Software Process Modelling and Enactment*. In: Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud co-located with the 17th

International Conference on Model Driven Engineering Languages and Systems (MODELS '14). Valencia, Spain. September, 2014.

STUDENT PAPERS

1. **S Alajrami** *On Cloud-Based Engineering of Dependable Systems*. In: Student Forum of the 10th European Dependable Computing Conference (EDCC '14). Newcastle upon Tyne, UK. May, 2014.

TECHNICAL REPORTS

1. **S Alajrami**, B Gallina, and A Romanovsky. *Enabling Global Software Development via Cloud-Based Software Process Enactment*. Technical Report TR-1494, School of Computing Science, Newcastle University, UK. March, 2016.

CONTENTS

1	Introduction	1
1.1	Preface	2
1.2	Software Engineering Evolution	3
1.3	Software Development as a Service	4
1.3.1	Motivation	4
1.3.2	Software process workflows	7
1.3.3	Initial experiments	10
1.3.4	Software development tools in the cloud	11
1.4	Thesis Storyline and Contributions	12
2	Reference Architecture for Software Development as a Service (SDaaS)	15
2.1	Introduction	16
2.2	Terminology & Definitions	16
2.3	Requirements for SDaaS	17
2.3.1	Non-cloud-related requirements	18
2.3.1.1	R1: Awareness and synchronisation support	18
2.3.1.2	R2: Availability of tools in real time	18
2.3.1.3	R3: Organisational policy convergence	19
2.3.1.4	R4: Capturing process and provenance data	19
2.3.1.5	R5: Accessible artefacts	20
2.3.1.6	R6: Governance and inter-organisation collaboration	20
2.3.2	Cloud-related requirements	20
2.3.2.1	R7: Privacy and legal compliance	20
2.3.2.2	R8: Multi-tenancy	21
2.3.2.3	R9: Scalability	21
2.4	Reference Architecture for SDaaS	21
2.4.1	WfMC compliance	21
2.4.2	Process modelling (Design Time)	24
2.4.3	The enactment service (Run-time)	24
2.4.3.1	Artefacts manager	24
2.4.3.2	External tools	25

2.4.3.3	The execution manager	25
2.4.3.4	Workflow engines registry	26
2.4.3.5	Scheduler	26
2.4.3.6	Consistency checker	26
2.4.3.7	SLA monitor	27
2.4.3.8	External workflow collaboration	27
2.4.4	Workflow engines	27
2.5	Specifications of the SDaaS Workflows	28
2.5.1	Activities types	28
2.5.2	Interaction patterns	29
2.5.3	Software workflows life-cycle	29
2.5.4	Activities life-cycle	30
2.5.5	Artefacts life-cycle	31
2.6	Proof of Concept	31
2.6.1	Implementation & deployment	32
2.6.2	Migrated tools	33
2.6.2.1	Spin	34
2.6.2.2	DiVinE	34
2.6.2.3	Concerto-FLA	35
2.7	Discussion	35
2.8	Summary	37
3	Modelling Software Processes for Cloud-Based Execution Using EXE-SPEM	38
3.1	Introduction	39
3.2	Background	40
3.2.1	Software process modelling	40
3.2.2	Software process modelling standards	42
3.2.2.1	SPEM2.0	42
3.2.2.2	ESSENCE	43
3.2.2.3	ISO 24744	44
3.2.2.4	Choosing SPEM2.0 for software process modelling	44
3.3	Requirements for Cloud-Based Executable Software Process Models	45
3.4	EXE-SPEM	48
3.5	Model to Text Transformation	51
3.6	Sample Process	53
3.7	Discussion	54
3.8	Summary	55

4	Cost-efficient Scheduling of Software Processes Execution in the Cloud	56
4.1	Introduction	57
4.2	Background	57
4.2.1	Workflow scheduling	58
4.2.2	Workflow scheduling algorithms	61
4.3	Scheduling SDaaS Software Workflows in the Cloud	65
4.3.1	Assumptions	65
4.3.2	Objectives	66
4.3.3	Motivation	66
4.3.4	Problem definition & assumptions	67
4.3.5	Scheduling requirements	70
4.3.6	Cost factors	71
4.3.7	Scheduling algorithms	72
4.3.7.1	Unlimited First Come First Serve (UFCFS)	74
4.3.7.2	Limited First Come First Serve (LFCFS)	74
4.3.7.3	Pool-based Adaptive Task Schedule	75
4.3.7.4	Proportional Adaptive Task Schedule	76
4.4	Evaluation	78
4.4.1	The request generator	78
4.4.2	The simulation scheduler	80
4.4.3	Workflow engines	80
4.4.4	Performing the simulation	81
4.4.5	Simulation results	81
4.4.5.1	UFCFS	82
4.4.5.2	LFCFS	83
4.4.5.3	Pool-based Adaptive Task Scheduling	84
4.4.5.4	Proportional Adaptive Task Schedule	85
4.5	Summary	86
5	Evaluation: A Case Study on Cloud-Based Engineering of Safety-Critical Systems Processes	89
5.1	Introduction	90
5.1.1	The evaluation method	90
5.1.2	The safety-critical systems case study	91
5.2	EXE-SPEM for Modelling Safety-related Processes	94
5.3	The PSSA Case Study	94

5.3.1	Argument generation	97
5.3.1.1	Product-based argument	97
5.3.1.2	Process-based argument	98
5.3.2	Implementation	101
5.3.3	Execution	103
5.4	Discussion	105
5.5	Summary	107
6	Conclusions	109
6.1	This Thesis in a Nutshell	110
6.2	Future Work	113
6.2.1	Motivating Scenarios	115
6.2.1.1	Continuous delivery	115
6.2.1.2	Compliance and continuous certification	116
6.3	Concluding Remarks	116
	Bibliography	118
	Appendices	128
A	The XML Schema for EXE-SPEM	129
B	The XML Process Model for Facebook’s Continuous Delivery Process	130
C	ARP4761	135
D	ARP4761 Wheel Brake System	140
E	Safety Case Representation	143
E.1	Visual representation	143
E.2	Machine-readable representation	144
E.3	Textual representation	144
F	The BSCU <i>Flamm</i> Architectural Model	147
G	The BSCU <i>Flamm</i> Architectural Model With FPTC Results	150
H	The Undesired Hazardous Events	154
I	The SACM/XMI Representation of the Product-Based Argument	155

J	The Argument Outline Textual Representation of the Product-Based Argument	156
K	The SACM/XMI Representation of the Process-Based Argument	159
L	The Argument Outline Textual Representation of the Process-Based Argument	162
M	The XML PSSA Process Model	165

LIST OF FIGURES

1.1	eSC workflow using the DiVinE activity	11
2.1	The WfMC workflow reference model components [67]	22
2.2	High level overview of the SDaaS reference architecture	23
2.3	The SDaaS reference architecture	25
2.4	Workflow life-cycle	30
2.5	Workflow instance life-cycle states	30
2.6	Activities life-cycle	31
2.7	Artefacts life-cycle	31
2.8	Message oriented communication	33
3.1	The process modelling components in the SDaaS reference architecture .	40
3.2	The meta-model of the extended SPEM 2.0 process structure	50
3.3	The meta-model of the XML format	52
3.4	Facebook's continuous delivery process model in EXE-SPEM	54
4.1	Highlighting the scheduler in the SDaaS reference architecture	58
4.2	The workflow scheduling categories [116]	59
4.3	Allocating activities to workflow engines (a)	74
4.4	Allocating activities to workflow engines (b)	74
4.5	Unlimited First Come First Serve algorithm	75
4.6	Limited First Come First Serve algorithm	76
4.7	Pool-based Adaptive task scheduling algorithm adapted from [109] . . .	77
4.8	The first workflow input model	79
4.9	The second workflow input model	79
4.10	The third workflow input model	79
4.11	Execution cost benchmark in all algorithms	82
4.12	Execution time benchmark for all algorithms for workflow 1	83
4.13	Execution time benchmark for all algorithms for workflow 2	84
4.14	Execution time benchmark for all algorithms for workflow 3	84
4.15	Execution times in UFCFS	85
4.16	Execution times in LFCFS for workflow 1	85

4.17	Execution times in LFCFS for workflow 2	86
4.18	Execution times in LFCFS for workflow 3	86
4.19	Execution time and cost benchmark in LFCFS	86
4.20	Execution times and cost in Pool-based Adaptive Task Schedule	87
4.21	Execution times in Proportional Adaptive Task Schedule	87
5.1	PSSA augmented with the argument generation process	95
5.2	The architectural model of BSCU components, ports and failures [2]	96
5.3	The pseudo code for analysing the FPTC results	98
5.4	Rules for product-based argument construction	99
5.5	GSN representation of the generated product-based argument	104
5.6	The product-based argument represented in text	104
5.7	GSN representation of partial process-based argument	105
5.8	A portion of the process-based argument represented in text	105
C.1	An overview of the airworthiness safety assessment process [2]	136
C.2	Example of the relationship between FHA and FTA [2]	138
D.1	The wheel brake system [2]	141
D.2	The wheel brake system sub-components [2]	142
D.3	FPTC syntax [54]	142
D.4	The architectural model of BSCU components, ports and failures [2]	142
E.1	GSN Elements	143
E.2	An example of a GSN argument	144
E.3	The GSN argument encoded in SACM/ARM XMI format	145
E.4	Example of the normal prose [68]	145
E.5	Example of the structured prose [68]	146
E.6	Example of the argument outline [68]	146
E.7	Example of the LISP style [68]	146

LIST OF TABLES

1.1	A sample of WfMSs	9
1.2	A sample of cloud-based software development tools	12
3.1	Comparing the three software process modelling standards	46
3.2	Graphical icons of EXE-SPEM elements	51
3.3	Mapping rules between EXE-SPEM and our XML notation	53
4.1	Workflow engine VM types and prices	80
4.2	Simulation results summary	82
4.3	Simulation results with confidence intervals	83
5.1	Claims to be validated using the case study	91
5.2	Concept mapping as proposed by MDSafeCer [56]	100

1

INTRODUCTION

Contents

1.1	Preface	2
1.2	Software Engineering Evolution	3
1.3	Software Development as a Service	4
1.3.1	Motivation	4
1.3.2	Software process workflows	7
1.3.3	Initial experiments	10
1.3.4	Software development tools in the cloud	11
1.4	Thesis Storyline and Contributions	12

An earlier version of some parts of this chapter is published in: S Alajrami, A Romanovsky, B Gallina: **Software Development in the Post-PC Era: Towards Software Development as a Service**. In: Proceedings of the 17th International Conference on Product-Focused Software Process Improvement (PROFES'16). Trondheim, Norway. November, 2016.

1.1 Preface

Software systems are playing a critical role in modern society. Many aspects of our lives such as transport, health care and communication are dependent on software. In a way, software is *smartifying* our lives through: smart phones, watches, glasses, grids and cities. The list goes on leading to a *smart society* where every aspect of the society is connected to, influenced by and dependent on software. Although, this helps addressing several societal challenges, it comes with the cost of increased software complexity. This complexity is then reflected on the way software is conceived where the expectations of reliability, security and fast delivery are higher than ever.

As Maximilien and Campos point out [86], we have entered the *Post-PC* era. This era is characterised by the increasing mobility and connectivity of people and devices, and the use of *cloud computing* as a software delivery platform. The role of the traditional personal computers (high-specification desktops) is gradually declining. Personal computers are becoming mobile and low-specification devices.

The software engineering community has long been evolving to address new rising challenges and to embrace new disruptive technologies. Market needs and economical factors create a challenge for software vendors to rapidly produce high quality software while maintaining low production costs. As a result, paradigms such as Agile methods, Continuous Delivery [70] and DevOps [78] were introduced.

Accordingly, the way software is conceived needs to adapt to the rising *Post-PC* era. Software development processes have two aspects: the software production aspect, which focuses on conceiving the software, and the management aspect, which focuses on planning and managing the development process [12]. Modern software is conceived by using a wide range of tools/platforms which support the software production aspect of software processes (development, testing, deployment and operation of software) as well as tools supporting the management aspect of software processes (project planning, resource planning, etc.). Some of these tools (e.g., IDEs like Eclipse Orion¹) are already being offered in the cloud. Chauhan and Ali Babar have proposed the Tools as a Service (TaaS) reference architecture [34]. TaaS focuses on provisioning tools supporting the software production aspect of software processes as services in the cloud, however, it

¹<https://orionhub.org/>

overlooks the management aspect of the development process. Such aspect is crucial for successful software projects. Therefore, there is a need for a solution which integrates the software production aspect (supported by tools) and the management aspect within the same environment.

In this thesis, we propose the *Software Development as a Service (SDaaS)* reference architecture which uses the cloud to support modelling, managing and enacting software processes in a model-driven paradigm. SDaaS utilises the cloud as an execution and distribution platform where tools are offered as services and orchestrated in workflows. Development environments are created on the fly and scaled as needed. Engineers can do their work on-the-go from anywhere. Furthermore, modelling and monitoring the process itself integrate the management aspect of the software development process into the development environment.

Throughout this thesis, we will be addressing some aspects of the SDaaS architecture in more detail. But first, let's go back in time and see how software engineering has evolved and how the challenges faced today have existed from the early days of software.

1.2 Software Engineering Evolution

From the early days of computing, production of software products has been challenged by various problems [96] some of which continued to persist as software manufacturing processes evolved.

The motivation behind establishing the foundations of the software engineering discipline was led by the continuous development of hardware technology such as processing, memory, storage, networking, which enabled producing larger and more complex software systems. The continuous growth of complexity meant higher risks of failures, where projects may run over budget and/or schedule. Both industry and academia needed to define a systematic way of engineering software systems which would minimise the risk of project failures. The challenge at the time was how to develop large and complex software in a systematic way within the available resources, such as money, time, and manpower, and it was known as the “software crisis” [41]. This led, for instance, to the NATO conference on software engineering in 1968 [89] which aimed to provide theoretical foundations and practical disciplines for software production just

like any other engineering discipline.

However, many would argue that the software crisis has never ended. Brooks has argued in [30] that there is no silver bullet in software engineering that can provide one order of magnitude of improvement. In his book [72], Jensen discusses why the issues that faced software in the 1960s are still plaguing software projects today.

The term *Software Processes* emerged in the 80s as the software industry realised the importance of software processes and their correlation with software quality [52]. Paulk et al. [92] describe a software process as “a set of activities, methods, practises, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases and user manuals)”. Research has focused on modelling, automation and improvement of software processes. Researchers have proposed various software process modelling languages (see Chapter 3) and software life-cycle models (e.g., Waterfall [95] and Spiral [28]) defining guidelines according to which software processes have to be carried out [52]. Furthermore, assessment and improvement of software processes were investigated.

Given that the challenges facing software development continue to exist, there is a continuous need to evolve the software development practises to cope with new technologies and economical challenges. In the next section, we describe our proposed approach for supporting software development in the *Post-PC* era.

1.3 Software Development as a Service

1.3.1 Motivation

In 1999, David Clark used the term *Post-PC* for the first time in a talk called “The *Post-PC* Internet”. He predicted that the future will be “inevitably heterogeneous” and “a network full of services”². Today, we are witnessing that era, where software and infrastructure are being delivered as services on the cloud. The cloud is becoming the development and the operation environment for software. This trend raises the need for alternative methods and technologies to design, implement, test, deploy and evolve software [53].

²<http://www.nytimes.com/1999/04/18/business/economic-view-is-mr-gates-pouring-fuel-on-his-rivals-fire.html>

Alternative methods need to consider the needs of modern software development. For example, modern software development does not recognise geographical/organisational boundaries. Global Software Development (GSD) [36] has crossed the geographical borders allowing teams around the globe to collaborate in distributed development projects. Furthermore, DevOps [78] is a trendy software development practice which aims at bridging the gaps between development and operations teams leading to an automated build-test-deploy-release cycle. Thus crossing the organisational boundaries. Additionally, in many cases, software development has to abide with certain standards or practises to ensure certain qualities in the produced software (e.g., safety).

The success of such projects (distributed, using trendy development practises and having high quality expectations) require the support for both aspects of the software development process; management and software production. Fuggetta and Di Nitto [53] point out that the software community is challenged with the need to move from rigid compliance to smart convergence. This means that the management aspect of the development process will be supported by monitoring and consistency checking tools. Such tools require information about the process, stakeholders involved and the process execution details (provenance data). In addition, with modern software development practises such as DevOps [78] and Continuous Delivery [70], the recommended practice is to increase automation of process steps whenever possible as it increases the productivity, supports repeatability and reduces errors. Therefore, the product development aspect of software processes should be supported with tool-chains which their execution can be (partially) automated.

In this thesis, we propose a reference architecture for **Software Development as a Service (SDaaS)**. SDaaS adopts the Model-Driven Engineering (MDE) [99] principles and leverages the cloud as an enabling platform for software process enactment (execution). It provides the core components to enable modelling and executing a software process in the cloud. Additional services and components can be built on top of the core SDaaS architecture to meet the needs of modern software development. The use of MDE principles aims at providing the basis for supporting the management aspect of software processes, while the use of the cloud aims at supporting provisioning of development tools for the software production aspect. In this thesis, we achieve support for the production aspect of the software development processes and the pave the

way for supporting the management aspect as well. SDaaS combines the following benefits of cloud computing and software process modelling:

Cloud benefits

- Cloud can be used to save resources (time, money, manpower) that are wasted in acquiring and configuring software development environments/tools. On-the-fly availability of development environments eliminates some manual configurations errors. Humble and Farley [70] point out that a simple error such as using different versions of the same tool by different teams can lead to expensive problems. Cloud-based development environments help software engineers to focus their efforts on the actual business problem rather than environments set up and configuration,
- The accessibility of the cloud facilitates collaboration between geographically distributed teams involved in software projects. In addition, it allows management to have a global view on the project progress. Artefacts can be globally managed and accessed. Group-oriented tools are identified by Boehm [29] among the needs that software engineering is required to achieve until 2025,
- Software development tools can be offered on demand in the cloud. This allows updating the tools without user involvement and also allows users to easily switch between different versions of a tool,
- By utilising the cloud economies of scale, computationally intensive tools (e.g., model checkers or provers) can be provided with sufficient computational resources as needed.

Software process modelling benefits

- Software process models (like all models) provide abstraction of process complexity [77]. Such abstraction is useful to hide unnecessary complexity from certain stakeholders and provide further details for others,
- Models hide plumbing details and can be used for communication and mutual understanding of processes between teams [17]. When process models are executable, it ensures that the processes will be executed similarly by different stakeholders or at different times,

- Executable process models can be monitored and tracked. This gives better global awareness and view for project managers especially in distributed projects,
- In certain domains (e.g., safety-critical systems), processes have to be strictly followed. Executable process models not only document and communicate the processes to be followed, but can also ensure that teams follow the organisation's policies and processes. Furthermore, they can be used as a form of evidence showing that a certain process has been followed.

The proposed SDaaS architecture is distinguished from the TaaS architecture [34] by two main features. First, while TaaS uses the cloud for provisioning tool-chains as a service supporting the software production aspect of the software development process, it completely overlooks the management aspect of the process. The SDaaS architecture uses software process models and capture process and execution data as well as artefacts which supports the management aspect of the software development processes, while it also uses the cloud to provision tools as services. Second, the SDaaS architecture provides granular control over the use of the cloud. For example, it allows to specify certain computational power or private hosting option for certain parts of the development process.

While the SDaaS architecture is generic and is aimed to be applied for any type of software development projects, the features it provides are particularly useful for GSD projects. Hashmi et al. [62] have already discussed the potential of using the cloud to facilitate GSD projects. The SDaaS architecture does not only utilise the cloud economies of scale, but also aims at supporting the management aspect of software development. In GSD projects, the challenges created by distances [32] require efficient management to overcome them.

1.3.2 *Software process workflows*

Software development process can be described as a sequence of operations (*activities*) performed by development team members including customers and managers (*actors*) where activities produce *artefacts* which are used as inputs for other activities. This complies with the Workflow Management Coalition (WfMC) definition of workflow [110] as “the automation of a business process, in whole or part, during which

documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". Therefore, software process can naturally be seen as a workflow. The idea of using workflow technology for software processes is not new, several researchers have investigated it [24, 33, 90]. Fuggetta [52] emphasises on the fact that software processes are processes too. Together with Di Nitto, they point out that reusing achievements from other domains (e.g., workflow management) for software processes has been overlooked by the software processes community [53].

Workflows are beneficial since they increase the visibility and automation of processes. Workflows also define the steps to be taken and the order of these steps. In addition, activities/tasks in a workflow can be assigned to certain actors (stakeholders) which allows for autonomous flow in the process while meeting all the necessary checkpoints. This allows management to focus on more strategic issues rather than focusing on daily tasks. Therefore, we follow the workflow approach for software process modelling and execution.

Software workflows are a special type of workflows and they differ from business and scientific workflows. First, they are often more complex [52] and interactive [53]. While some business workflows may include collaborative teamwork, once the workflow is designed and finalised, it will not be changed frequently and the involved stakeholders will be doing what the process prescribes most of the time. Software workflows are more dynamic and subject to frequent change. Additionally, software workflows might change from one project to another. Scientific workflows, on the other hand, are less interactive and contain less control flow.

Existing Workflow Management Systems (WfMSs) are designed for either business or scientific applications. Table 1.1 describes a sample of WfMSs and their strengths and weaknesses from the perspective of supporting software workflows in the cloud.

These WfMSs are either commercial or open source tools. While some of them use the cloud as an infrastructure, many do not. Those using the cloud (e.g., eScience Central) lack fine-grained cloud-based execution configurations. A common limitation across all of them is the lack of support for modelling software processes using any standardised software processes modelling language (e.g., SPEM2.0 [5]). In the next

³<http://www.bonitasoft.com/>

⁴<http://www.runmyprocess.com/>

Table 1.1: A sample of WfMSs

WfMS	Strengths & Weaknesses
<i>Bonita</i> ³	Strengths: - supports creating BPMN [7] processes, - provides a shared repository of processes, - can connect with external systems.
	Weaknesses: - not cloud-based, - does not support software process modelling languages, - does not support software development tools on the fly.
<i>eScience Central</i> [66]	Strengths: - cloud-based (SaaS application), - supports customised executable blocks building in different languages, - collects provenance data, - facilitates collaboration between scientists.
	Weaknesses: - supports scientific (data-driven) workflows but not software process (process-driven) workflows, - does not support software process modelling languages, - does not support configuring individual workflow activities' execution.
<i>Taverna</i> [112]	Strengths: - has a desktop workbench and command line tool, - has a web-based version of the workbench, - supports external services through WSDL/SOAP.
	Weaknesses: - supports scientific (data-driven) workflows but not software process (process-driven) workflows, - does not support software process modelling languages.
<i>Yawf</i> [105]	Strengths: - supports business process models using BPMN, - uses Petri-Nets to capture control-flow processes, - has a formal foundation and supports exception handling, - has a service oriented environment for workflow execution.
	Weaknesses: - does not support software process modelling languages.
<i>RunMy Process</i> ⁴	Strengths: - cloud-based (uses AWS for infrastructure), - supports creating custom business processes and deploys them in the cloud, - provides connectors to connect with SaaS and on-premise applications.
	Weaknesses: - does not support software process modelling languages.

subsection, we demonstrate the initial experiments which we ran on eScience Central to examine its suitability for executing software processes.

1.3.3 Initial experiments

We wanted to check if the existing WfMSs can be used to support modelling and executing software development processes. We conducted initial experiments on eScience Central (eSC) [66] to model and execute simple model checking processes. eSC is a science as a service platform which supports cloud-based execution, drag, drop and connect supported workflow activities (called boxes in eSC), building workflow activities in different languages (e.g., R⁵ and Java⁶), storing artefacts in the cloud and collecting provenance data about workflow execution. Because of these features, eSC was the closest match (out of the WfMSs that we have reviewed in Table 1.1) for cloud-based execution of software processes.

The objective of this initial experiment is to assess if eSC can be used to: (a) model software process models and (b) execute these models in the cloud. We integrated two model checking tools into the eSC platform and created workflow activities wrapping them. The model checkers are: Spin [59] and the distributed model checker DiVinE [23]. The Spin-based model checking activity was configured to accept an input model and perform the model checking then generate the results. The DiVinE-based model checking activity was developed to accept input model and configurations of how the distributed model checking task should be executed. These configurations include: the number of virtual machines (VMs) to be used and a timeout after which the activity automatically scales the number of VMs either exponentially or linearly. Figure 1.1 shows the workflow created in eSC using the DiVinE-based model checking activity.

We were able to execute this workflow in the cloud and produce the model checking results. However, when we want to guide the workflow execution while it is executing (e.g., to decide if the workflow should scale resources up and continue execution in the case of reaching timeout), it is not possible because eSC does not support interactive activities. Additionally, eSC does not support modelling control flow elements such as *loops*. We embedded the utilisation of the cloud for the distributed execution in the

⁵<https://www.r-project.org/>

⁶<https://www.java.com/en/>

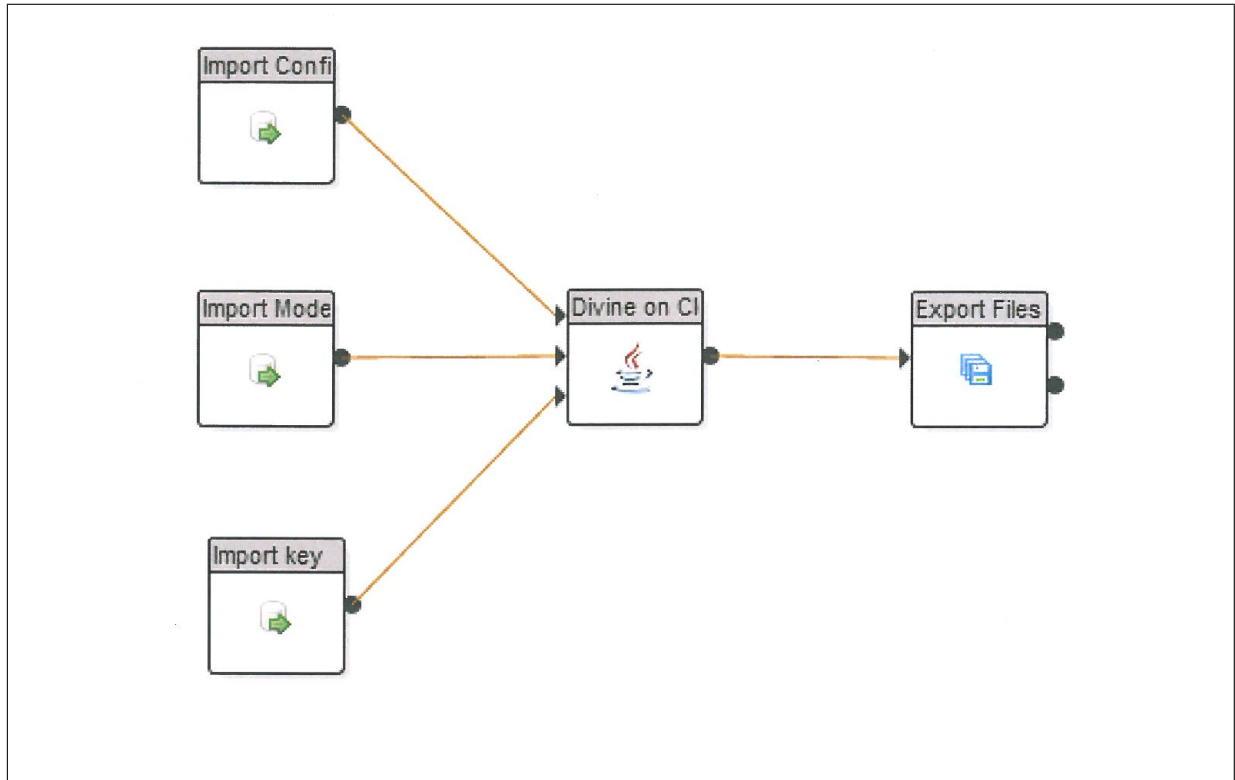


Figure 1.1: eSC workflow using the DiVinE activity

DiVinE-based model checking activity since the eSC platform did not allow configuring the computational power required for each workflow activity. Finally, we are not able to use any standard software process modelling language. Therefore, we conclude that we need a new platform to support our SDaaS vision.

1.3.4 *Software development tools in the cloud*

Using the cloud as a platform for development is not new. Several development tools are already offered in the cloud. Table 1.2 provides a list of some academic and industrial cloud-based tools. The list provides a sample of tools and is not meant to be comprehensive. As we can notice, despite that some of these tools provide a rich set of features, none of them supports the entire Software Development Life-cycle (SLDC) and none of them supports capturing and executing customised software processes.

⁷<https://codenvy.com/>

⁸<https://orionhub.org/>

⁹<https://cloud-playground.appspot.com/playground/>

¹⁰<https://github.com/>

¹¹<http://cloudforge.com>

¹²<http://www.ibm.com/cloud-computing/bluemix/>

Table 1.2: A sample of cloud-based software development tools

Tool	Description
<i>Codenvy</i> ⁷	Provides cloud workspaces for development projects. It allows creating docker containers, supports team collaboration and can be deployed on a private cloud. However, no support for process capturing or monitoring nor for provenance data collection.
<i>Eclipse Orion hub</i> ⁸	A cloud-based editor that allows to code, deploy and run source code in the cloud. Does not have any support for external tools nor for process capturing and monitoring. Only covers the development phase.
<i>Google cloud playground</i> ⁹	An experimental development platform which aims to give developers a hands-on experience in developing software in Google Appengine. Only serves as an educational tool with basic editing features.
<i>Github</i> ¹⁰	A source code management tool which allows developers to collaborate on source code projects. Limited to source code management and does not support capturing or monitoring software development processes.
<i>CloudForge</i> ¹¹	A development platform which supports project and team management and project deployment to public/private clouds. It does not capture or monitor the development processes nor support all phases of the SDLC.
<i>Cloud9</i> [31]	Provides a parallel symbolic-execution-based testing platform as a service on the cloud. Only supports testing and not the whole SDLC.
<i>Yeti</i> [91]	An academic automated random testing tool deployed in the cloud. Only supports testing and not the whole SDLC.
<i>IBM Bluemix</i> ¹²	A platform for building and deploying software projects by utilising existing open source tools. However, it does not support capturing or monitoring the software development processes.

1.4 Thesis Storyline and Contributions

Cloud computing has evolved to become the enabling platform for the Post-PC era applications. Not only cloud became the deployment environment due to its economies of scale but also it became the development environment [53]. By observing research publications related to software engineering for/in the cloud (e.g., the journal of software and systems special issue [19] and the IEEE Services Track [18]), we conclude that

research (e.g., [11, 100]) has been focusing on adapting software engineering practises to fit for the types of applications built in the Post-PC era. This thesis, however, focuses on utilising the cloud for software development. More specifically, utilising the cloud for software process enactment.

We envision to leverage the benefits of MDE and cloud computing -as discussed in Section 1.3- for supporting both the software production and the management aspects of software processes. As we explained in Section 1.3.2, software processes models are naturally workflows. Consequently, we use workflow systems to manage the execution of software process models. Our initial experiment -as demonstrated in Section 1.3.3- has shown that existing WfMSs are not suitable for modelling and executing software processes in the cloud. *To the best of our knowledge, this thesis is the first study on cloud-based software processes execution.*

This thesis makes the following contributions:

- We propose a reference architecture for Software Development as a Service (SDaaS) [15]. The SDaaS architecture enables modelling cloud-based executable software processes and enacting them in a hybrid cloud. Combining the model-driven approach with the cloud allows for fine-grained configurable execution which uses the cloud's elasticity, availability and accessibility. In addition, SDaaS facilitates distributed development through shared and accessible process models and artefacts. The SDaaS architecture is described in detail in Chapter 2,
- We extend the OMG Software and Systems Process Engineering Meta-model (SPEM2.0) to allow modelling cloud-based executable processes. The extended version is called *EXE-SPEM* [13]. It allows modelling cloud-specific requirements such as computing power and privacy levels. We also introduced an XML schema to map the graphical EXE-SPEM models into a machine consumable XML representation. EXE-SPEM is described in detail in Chapter 3,
- We propose the Proportional Adaptive Task Schedule algorithm to schedule software workflows execution. The algorithm aims at reducing the execution cost without significantly increasing the execution time (makespan). It uses the workflow models to predict upcoming load on hourly bases and compares it with historical load for the past hour then dynamically scale computational resources

up or down as needed. We show through simulation that this algorithm saves between 19.74% and 45.78% of the execution cost when compared to three other adapted algorithms while providing the best resource utilisation amongst them and the second best execution time,

- As an initial evaluation, we instantiate the SDaaS reference architecture and implement a proof-of-concept which supports a subset of the SDaaS features. Additionally, we integrate three different tools in this proof-of-concept implementation. This implementation proves the feasibility of the SDaaS architecture and is discussed in detail in Chapter 2,
- We evaluate our vision by using the implemented proof-of-concept to support executing safety-critical systems processes [14]. We model a safety-related process adopted from the aerospace domain standard; ARP4761 and use the proof-of-concept SDaaS implementation to execute it. The evaluation aims to show: the applicability of our vision for development processes, how using the cloud and process models can open new opportunities and how it can save cost and time. We show that by using the process model and the provenance data collected by the SDaaS proof-of-concept we are able to automatically generate safety case argument fragments. We describe this case study in Chapter 5.

In the next chapter, we describe the SDaaS reference architecture.

2

REFERENCE ARCHITECTURE FOR SOFTWARE DEVELOPMENT AS A SERVICE (SDAAS)

Contents

2.1	Introduction	16
2.2	Terminology & Definitions	16
2.3	Requirements for SDaaS	17
2.3.1	Non-cloud-related requirements	18
2.3.2	Cloud-related requirements	20
2.4	Reference Architecture for SDaaS	21
2.4.1	WfMC compliance	21
2.4.2	Process modelling (Design Time)	24
2.4.3	The enactment service (Run-time)	24
2.4.4	Workflow engines	27
2.5	Specifications of the SDaaS Workflows	28
2.5.1	Activities types	28
2.5.2	Interaction patterns	29
2.5.3	Software workflows life-cycle	29
2.5.4	Activities life-cycle	30
2.5.5	Artefacts life-cycle	31
2.6	Proof of Concept	31
2.6.1	Implementation & deployment	32
2.6.2	Migrated tools	33
2.7	Discussion	35
2.8	Summary	37

An earlier version of some parts of this chapter is published in: S Alajrami, A Romanovsky, B Gallina: **Software Development in the Post-PC Era: Towards Software Development as a Service**. In: Proceedings of the 17th International Conference on Product-Focused Software Process Improvement (PROFES'16). Trondheim, Norway. November, 2016.

2.1 Introduction

Software development approaches and tools evolve to meet the continuously rising demand for quality (e.g., reliability, safety and security) of software systems. A typical modern software development would require the use of multiple tools and platforms to support the different phases of the development life-cycle. Such tools are provided by different vendors and are often not interoperable. Furthermore, the management of the development process is not inherently integrated in these tools and is performed as a separate process. As explained in Chapter 1, we need alternative development methods which integrate the process management perspective and support using tools on the fly. An approach which utilises the benefits of the cloud (e.g., accessibility and elasticity) and model-driven engineering (e.g., different levels of abstraction) for software development processes.

In this chapter, we propose the Software Development as a Service (SDaaS) reference architecture. SDaaS supports modelling, enacting and managing software processes. Rather than focusing on a particular process model (e.g., waterfall or spiral), SDaaS supports any software process model. In the next section, we define some terms and assumptions which are used throughout this thesis.

2.2 Terminology & Definitions

Before we dive into designing the reference architecture for SDaaS, we need to define the terms that will be used/mentioned throughout this thesis. These terms are defined below in the context of the SDaaS architecture and incorporate some assumptions that we have made.

Definition 1 *Software Process* is defined as “a set of activities, methods, practises, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals)” [92]. Such process would involve the use of various tools and covers both the management and the technical aspects of software development.

Definition 2 *Process Author* is defined as “the actor who models a software process”. The process may have one or more authors and can be enacted by users other than

those who authored it. Synonyms include: process engineer.

Definition 3 *Software Workflow* is defined (by adapting the Workflow Management Coalition (WfMC) definition of *Workflow* [110]) as “the automation of a software process, in whole or part, during which artefacts, information or activities are passed from one participant to another for action, according to a set of procedural rules”.

Definition 4 *Activity* is defined as “the smallest unit for breaking down workflow tasks”. Activities can be assigned to one or more actors. Synonyms include: task.

Definition 5 *Artefact* is defined as “any object which is used as an input for a software process or produced during the enactment of its activities”.

Definition 6 *Tool* is defined as “any computerised program that is used to assist carrying out the activities of a software process”.

Definition 7 *Actor* is defined as “any person who is involved in creating, modelling, influencing and enacting a software process. Examples include: developers, customers, managers, etc.”. Synonyms include: user, stakeholder.

Definition 8 *Software Process Enactment* is defined as “the act of executing all the individual activities of a software workflow instance by actors and with the help of tools”. It is worth noting here that we do not define enactment as just monitoring and tracking a process (e.g., [45, 93]) but as the actual execution act of the process and its activities.

2.3 Requirements for SDaaS

In this thesis, we focus only on the technical aspects of the SDaaS architecture. The business and economical perspectives of the service are, although important, beyond the scope of this thesis. In this section, we elicit the list of requirements that the SDaaS architecture should meet. These requirements are identified by observations that we made during our initial experiments which we discussed in Section 1.3.3. In these experiments, we attempted to execute software processes in the cloud using an existing workflow management system (eSC [66]) and identified the following list of requirements as being missing from existing workflow management systems and essential to enable cloud-based execution of software processes. Since this list of requirements is drawn from our experiments, we do not consider it complete.

2.3.1 *Non-cloud-related requirements*

2.3.1.1 R1: Awareness and synchronisation support

Globalisation of markets increases the distribution of software development. In such cases, coordination is needed to manage dependencies between development tasks. Geographical and temporal distances in distributed development impair the coordination between distributed teams. Herbsleb [65] argues that such impairment is the result of *less communication, lack of awareness and incompatibilities*. He argues that distributed teams often have little shared context which leads to lack of awareness and misunderstandings about what other teams are doing. Dourish and Bellotti [43] describe group awareness as “an understanding of the activities of others, which provides a context for your own activity”. Lack of awareness hinders the project management as it makes it difficult to track changes as they propagate between distributed locations [65]. Furthermore, Gutwin et al. [60] argue that lack of awareness is also responsible for problems such as duplicate work, overwritten changes and incorrect assumptions between team members. In order to avoid misunderstandings and misalignment between teams, it is required to keep the distributed teams and their management aligned and aware of the overall project progress. A unified and accessible development platform where everyone can be aware of the process being followed, the tools being used and the overall progress of the project would be helpful to enhance awareness and synchronisation between teams. Better awareness of what is going on in a project can help managers to detect and mitigate challenges in distributed development [80].

2.3.1.2 R2: Availability of tools in real time

As mentioned in the previous subsection, Herbsleb [65] listed *incompatibilities* as one of the causes of coordination impairment. He refers to incompatible tools and processes across distributed locations. Chauhan and Babar [34] list several advantages of offering tools as a service (TaaS) in the cloud. The main benefits include: support for awareness and alignment of tools with processes. Acquiring software development on the fly saves time and cost for setting up, configuring and maintaining your own environment. Similarly, to keep the focus on the business problem, tools should be available as services which can be accessed on demand with flexible pricing models. This also makes it

easier to guarantee that all teams are aligned in the tools (and versions) they are using. Incompatibilities in development tools can lead to costly problems (e.g., see [70]). Additionally, expensive commercial tools can be made available on a pay-as-you-go model.

2.3.1.3 R3: Organisational policy convergence

In certain cases, certain policies/standards have to be followed for quality assurance or certifications purposes. An example is the safety-critical systems development. Such development need to follow stringent processes defined in standards to ensure the safety of the produced (software) system. Another example, is enforcing certain practises and cultures within development teams to ensure quality. Such practises include, for instance, frequent code commits and using continuous integration in an agile development project. However, Fuggetta and Di Nitto [53] call for smart convergence rather than rigid processes. Therefore, it should be possible to allow stakeholders to flexibly define their process models and monitor their compliance with the standard-/recommended process.

2.3.1.4 R4: Capturing process and provenance data

Today, there is no need to emphasise the importance of data. Both research and industry are pushing the limits to collect more data, reason about it and process it faster. Software development data is no exception. Capturing data about the software process execution (provenance data) such as artefacts, versions, time, people involved, etc. can be useful for different purposes, particularly, to support accountability and traceability. For example, in safety critical systems, safety cases include evidence to describe the process being followed in order to show that it has met the certification standards [14]. Such evidence can be supported by provenance data collected during the process execution. Process improvement is another example where provenance data can be analysed to find weakness areas (e.g., [38]). Therefore, provenance data need to be collected, stored in tractable form and ready to be exploited [65].

2.3.1.5 R5: Accessible artefacts

Artefacts are an important part of software processes. Artefacts include: source code, requirements, documentation, tests, configurations, etc. As these artefacts evolve throughout the development process, they accumulate valuable information about the process. However, inconsistencies also accumulate [16]. Therefore, artefacts should be stored and maintained in an accessible and traceable way by all authorised stakeholders regardless of their locations. Changes should be captured in different versions of artefacts and each version should be associated with meta-data that can be used for traceability.

2.3.1.6 R6: Governance and inter-organisation collaboration

Facilitating outsourcing of parts of the software development processes to sub-contractors while ensuring privacy and confidentiality of data and processes is essential in the modern software industry. Companies should be able to host the software development process of their sub-contractors on their private cloud infrastructure while giving them access to the artefacts they need. This eliminates the risks associated with sending private confidential artefacts outside of the company's network. The *Software outsourcing* scenario in Chapter 1 demonstrates such situation which is inspired from a real industrial context.

2.3.2 *Cloud-related requirements*

2.3.2.1 R7: Privacy and legal compliance

Using the cloud raises concerns about privacy and security [114]. As the software development will take place on the cloud provider's infrastructure, companies may not be willing to put confidential artefacts on public clouds. Similarly, regulations may impose restrictions as to where processes can take place. For example, the EU regulations [48] impose that all EU data should be processed and stored within the EU unless companies can demonstrate that they take sufficient measures to protect the EU users privacy and data. Therefore, there is a need for the process execution to be configurable and take place on a hybrid cloud (private and public) when needed.

2.3.2.2 R8: Multi-tenancy

Multi-tenancy is a key characteristic of cloud applications [50]. It refers to having multiple tenants sharing the cloud service (software, platform and/or infrastructure). The SDaaS architecture is meant to support multiple stakeholders involved in different software projects. Therefore, it should provide a multi-tenant platform.

2.3.2.3 R9: Scalability

Cloud offers flexible scalability of computational resources. In the context of the SDaaS architecture, scalability can be interpreted in two dimensions: a) scalability of computational resources, and b) scalability of the number of supported stakeholders (tenants) and software projects. The SDaaS architecture should be able to scale both the computational resources and the number of stakeholders on demand.

2.4 Reference Architecture for SDaaS

In this section, we propose a reference architecture to support Software Development as a Service (SDaaS). This architecture is designed to be general and to be applied to any type of process models (e.g., agile, waterfall, etc.). Consequently, we present some high level description of some of the architecture components here without exploring in depth all the possibilities it offers. Since we model software processes as workflows, the SDaaS architecture is a Workflow Management System (WfMS). The next subsection explains how the SDaaS architecture complies with the WfMC reference model.

2.4.1 WfMC compliance

Historically, WfMSs had taken different typologies [67, 87]. For instance, some relied on circulating documents between participants (*Document-centric*) while others used an email-system to pass messages around (*Email-based*). The variety of modelling formats and types of WfMSs has raised the need for standardisation and that is where the Workflow Management Coalition (WfMC) fits in. WfMC aims to standardise workflow management systems to allow interoperability between them. For this purpose, WfMC has proposed the workflow reference model [67]. Compliance with this model means that a WfMS can interact with other WfMSs.

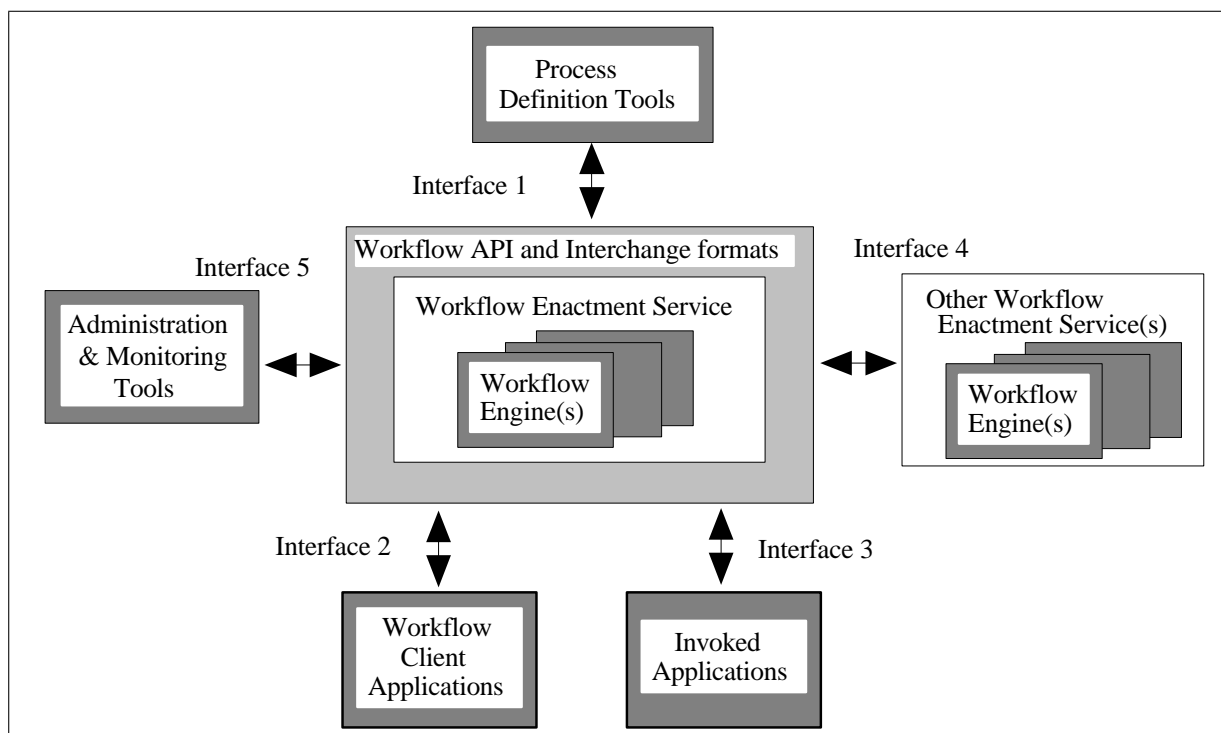


Figure 2.1: The WfMC workflow reference model components [67]

In this subsection we describe the WfMC reference model and show how our proposed SDaaS architecture complies with it. The reference model defines at the high level the architecture of a WfMS as illustrated in Figure 2.1.

In the model, the **Process Definition Tools (Interface 1)** refer to the tools supporting creation and definition of processes. The **Workflow Client Applications (Interface 2)** allow users to interact with the WfMS and receive work items (activities). As actors perform their assigned activities, they may need to invoke external applications which is facilitated by the **Invoked Applications (Interface 3)**. To support interoperability with other WfMSs, the **Other Workflow Enactment Services (Interface 4)** enable integration between different WfMSs. The **Administration & Monitoring Tools (Interface 5)** provide workflow management and monitoring control features. The five interfaces are connected with the **Workflow Enactment Service** through the **Workflow API (WAPI)**. The workflow enactment service contains one or more **Workflow Engines** which provide the run-time execution environment for a workflow instance.

We design the SDaaS reference architecture to comply with WfMC reference model (i.e., to provide the same type of interfaces). The WfMC reference model was proposed in the 1990s with centralised workflow systems in mind (i.e., where workflows are executed on a single machine). In the SDaaS reference architecture, we adapt the enactment

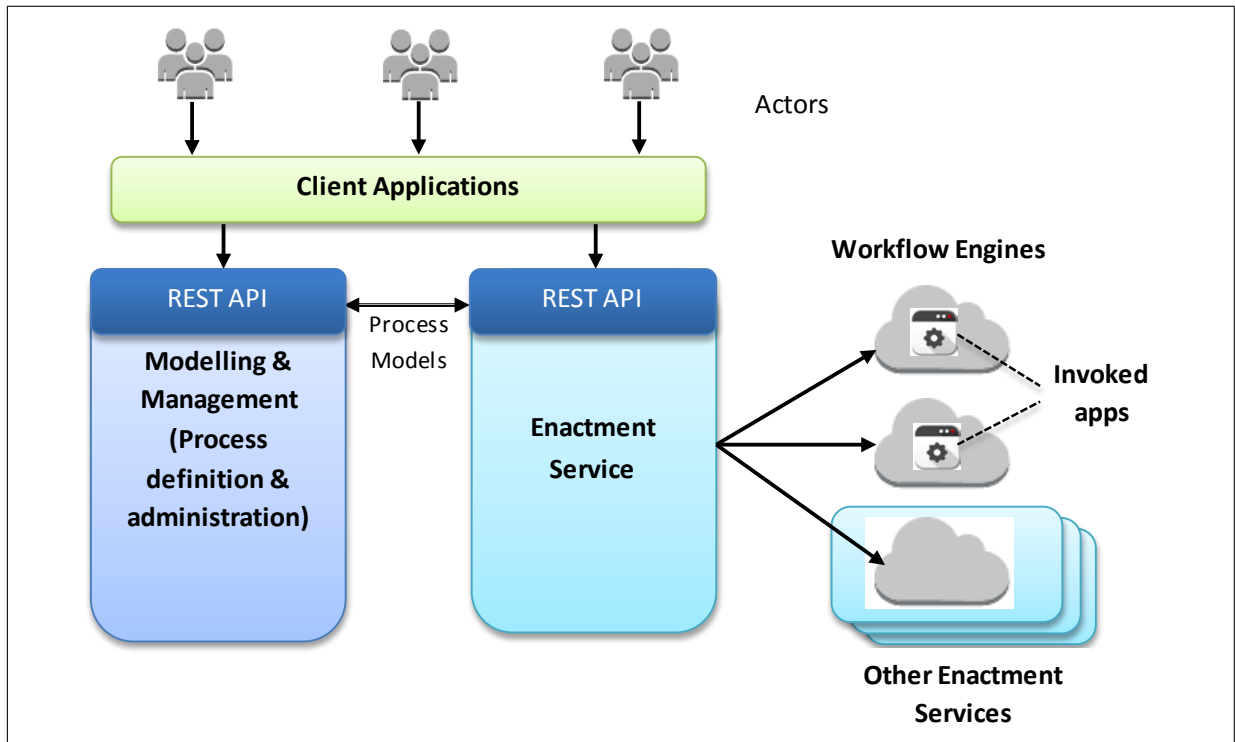


Figure 2.2: High level overview of the SDaaS reference architecture

service to support distributed cloud-based software process execution.

Compliance with the WfMC reference model ensures that the implementations of the SDaaS reference architecture can interoperate with other workflow systems. Figure 2.2 shows the high level overview of the SDaaS reference architecture which consists of four main components. The *Client Applications* can be any desktop, web or mobile applications through which users interact with the SDaaS architecture. This represents interface 2 from the WfMC model. The client applications interact with both the *Modelling & Management* and the *Enactment Service* components through their individual REST APIs. The *Modelling & Management* component combines interfaces 1 and 5 from the WfMC model. The *Enactment Service* executes workflows on distributed workflow engines and manages the invoked applications and interactions with other workflow enactment services (Interfaces 3 and 4).

Figure 2.3 provides a detailed view of the SDaaS reference architecture while the next subsections provide details about its components.

2.4.2 *Process modelling (Design Time)*

The software process design time is the phase where the software process model is created. In programming terms, it is similar to writing code before (compiling and) executing it. Software process modelling and the modelling language that we use (EXE-SPEM) will be covered in Chapter 3.

As shown in Figure 2.3, the software process design time components include: a) the **Process Model Authoring** module which allows constructing process models using EXE-SPEM constructs, b) the **Process Access & Sync. Service** which applies access management policies and ensures the consistency of models that are being authored by distributed teams simultaneously. This is done by applying appropriate read/write locks. This module also notifies collaborators when a model is changed/updated, c) the **Process Model Storage Service** which allows saving the model into the cloud-based repository through the REST API. The model can be retrieved for editing/enactment using the same module, and finally, d) the **Process Model Transformations** module which transforms models into the executable XML notation from EXE-SPEM (or potentially other modelling notations). This module contains an adapter for each possible transformation .

2.4.3 *The enactment service (Run-time)*

The enactment service interacts with the process modelling service through the REST APIs. Behind the API, the service is responsible for the run-time instantiation and execution of process models. To do this, the service consists of several modules as illustrated in Figure 2.3. These modules are:

2.4.3.1 **Artefacts manager**

Software processes involve producing large number of artefacts such as code, models, test cases, requirement documents, documentation, etc. These artefacts capture invaluable information about both the software process and product evolution. The artefact manager stores the artefacts themselves and meta-data about them into the artefacts repository. The meta-data include: actors involved, version, tools used and the date and time on which the artefact was created/modified. It is worth noting that process

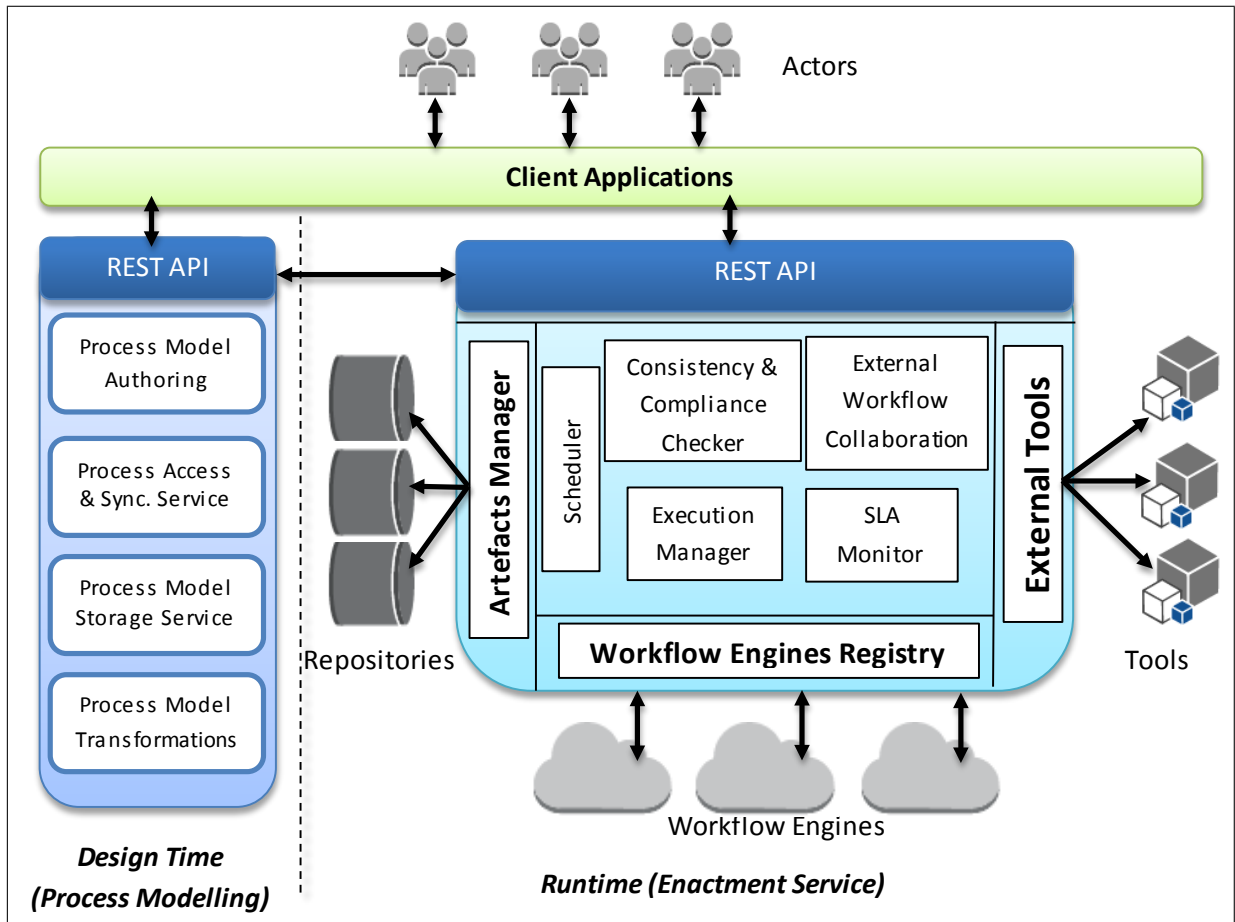


Figure 2.3: The SDaaS reference architecture

activities (wrapping development tools) are also treated as artefacts and are stored in the artefact repository. Although mining the artefact repository is out of the scope of this thesis, several approaches to mining software artefacts exist in the literature. Some of which have been surveyed by Kagdi et al. [73].

2.4.3.2 External tools

External tools are service blocks performing the software process activities. These blocks are either: interactive, control points (providing control flow during the process execution) or automated fire-and-forget activities. This module provides the necessary information to trigger compatible external tools and handles the interaction with them. Compatible tools are offered as a service and can be triggered through service calls.

2.4.3.3 The execution manager

The execution manager orchestrates the enactment of process models. First, an instance of the model is created and the ready-to-execute activities are passed to the scheduler

to be scheduled. The scheduled activities are then executed on workflow engines. During the execution of the process, the execution manager tracks the status of the process instance being executed. The status goes from *Inactive* to *Active* and terminates with a *Completed* state. This module also logs all the provenance data about each process instance execution. Furthermore, the execution manager handles interactions with stakeholders when executing an interactive activity (see Section 2.5.1 for different types of activities).

2.4.3.4 Workflow engines registry

The Workflow Engines Registry is responsible for starting, stopping and monitoring workflow engines based on the scheduling policies used by the scheduler (workflow scheduling will be discussed in Chapter 4). Workflow engines are independent applications running on different cloud providers. Activities get executed in a workflow engine that is deployed on a public or a private cloud. The workflow engine has to meet the execution requirements expressed in the process model. The execution of activities is a black-box execution which means that the workflow engine would not know any information about the process being executed. This reduces the risks of privacy and confidentiality breaches.

2.4.3.5 Scheduler

The Scheduler handles the scheduling of process activities' execution. This involves checking the required computational resources (from the process model) and allocating activities to suitable workflow engines. The scheduler operate using a policy to meet the the enactment requirements (e.g., enacting an activity on a private cloud) while minimising the cost. The schedules generated by the scheduler determine the expected load of execution and is used by the workflow engines registry to dynamically scale the number of workflow engines. Scheduling will be covered in detail in Chapter 4.

2.4.3.6 Consistency checker

As explained in Section 2.3, the automated consistency checking for the process during its execution can alleviate development problems (e.g., deviating from a standard process) early and save time and cost. Some approaches check if process models

are consistent with the requirements [63] or if the implementation is consistent with the architectural description [26]. Such checks rely on data and models. The SDaaS architecture maintains the process models and other artefacts (e.g., code) in the artefact repository. Additionally, it records provenance data about the process execution. This data can be used for consistency analysis to provide similar results to the studies mentioned earlier or even to pave the way for more comprehensive checks. However, the formulation of consistency checking rules are beyond the scope of this thesis.

2.4.3.7 SLA monitor

As explained in the software outsourcing scenario (see Chapter 1), when two or more organisations collaborate on a project, SLA monitoring becomes handy to transparently ensuring that all parties are not breaching the SLA. While each organisation can have its own SDaaS environment, these environments can exchange data about the process state and execution using the *external workflow collaboration* module.

2.4.3.8 External workflow collaboration

The External Workflow Collaboration allows process execution to incorporate invoking processes managed by another workflow system (e.g., from a different organisation). The invocation is done through service calls. This enables business collaborators to have a global view of the project without interfering or accessing the internal processes of each other.

2.4.4 Workflow engines

Workflow engines are the execution containers for executing activities and they can be deployed on any public or private cloud. Workflow engines register themselves with the enactment service when they start, which allows adding more workflow engines dynamically. Activities are allocated to a particular workflow engine by the scheduler of the enactment service. Once a job has been received, the workflow engine requests the resources (artefacts and executables) required to execute this activity from the enactment service through the REST API. The workflow engine updates the enactment service with the execution progress throughout. When the execution is finished, the

workflow engine uploads any produced artefacts to the enactment service and performs a clean up which leaves no traces of this execution on the workflow engine.

2.5 Specifications of the SDaaS Workflows

Our approach to define SDaaS is based on treating software processes as workflows. Such workflows consist of several types of elements. In this section, we describe the main elements and their life-cycles from the time of creation until the workflow is fully executed. Software workflows consist of a set of *Activities*, *Artefacts* and *Actors*. Modelling of these elements will be discussed in Chapter 3.

2.5.1 *Activities types*

Software workflows are complex and contain various types of activities performed by different actors. In general, these activities can be categorised into three categories:

- **Automated Activities.** Some activities in a software process can be supported by automated tools. Such tools will need to receive an input (in the form of artefacts and/or parameters). Examples of such activities include different types of testing, model checking and data analysis. Depending on the size of the inputs (e.g., test cases, models to verify, etc.) such activities need to handle, they might be computationally intensive and take long time to finish executing.
- **Interactive Activities.** Unlike the automated activities, interactive activities require the involvement of actors to make decisions, provide input or perform manual tasks such as editing code and models.
- **Control Points.** At certain points in the software workflow, there will be a need for making decisions about the next steps in the workflow enactment. The decision could be to go back and change the process or the input artefacts or going into one of multiple possible forward paths in the workflow. For example, based on testing results, the actor can decide whether the code still need to be modified or not.

2.5.2 *Interaction patterns*

There are two possible types of human-computer interactions which take place in software workflows. The first is the interaction to create, edit, manage and enact software workflows. This can be achieved using the user interfacing tools (aka workflow client applications). Such tools can be desktop-based, web-based or mobile applications. The other type of interaction is the interaction during the execution of the software workflow. As mentioned earlier, software workflows contain different types of activities. Some of these activities are interactive where actors need to interact with the workflow to provide instructions, decisions, input parameters and to edit artefacts. These activities need to have a way of sending messages and receiving responses from the relevant actors. Long interactions such as editing code or other artefacts can be done offline using the workflow client applications where the workflow enactment will be paused till the editing is finished. It is worth noting that we do not investigate the human-to-human interactions between actors in this thesis.

2.5.3 *Software workflows life-cycle*

In our vision, software workflows are executable and live throughout the development process. Software workflow models represent software processes. As shown in Figure 2.4, they come to life after process author(s) construct(s) the workflow model using a software process modelling language (which will be discussed in Chapter 3). After being created, the workflow models can be instantiated as many times as necessary and those instances can then be enacted. Enacting a workflow model will result in the generation of new artefacts as prescribed in the model. If the workflow model needs to be adapted (e.g., for another project or to improve the process or use a different tool), the model is modified, then new instances can be created and enacted.

Software workflow instances move between three states as shown in Figure 2.5. Initially, the instance is *Inactive* and once it starts to be enacted, it becomes *Active*. After completing the execution of an instance, the state becomes *Completed*. The completed state indicates that the execution of an instance has terminated either successfully or not.

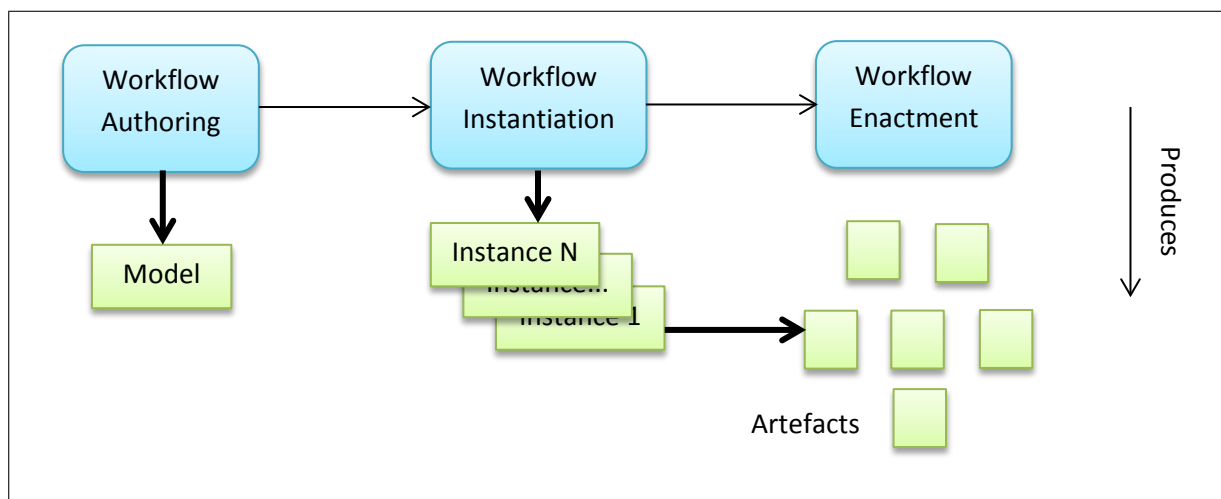


Figure 2.4: Workflow life-cycle

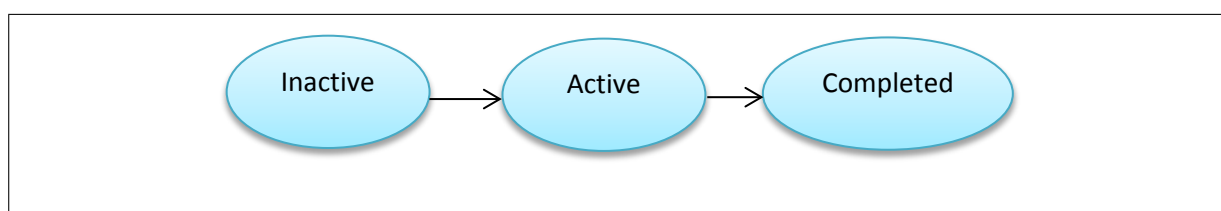


Figure 2.5: Workflow instance life-cycle states

2.5.4 Activities life-cycle

Software workflows consist of a set of activities. These activities are executable and they can be custom-made tools/applications or standard CASE tools. They are integrated to the SDaaS architecture (stored in the artefact repository) and are executed when a workflow instance containing them is enacted. During the workflow instance enactment, the activities status change between five states. As we can see in Figure 2.6, an activity is initially in the *Inactive* state which means that it has not become ready to execute yet. The *Ready* state indicates that the activity is now ready to be executed which means that all its preconditions (availability of input artefacts/parameters) have been met. Once in the Ready state, the scheduler of the enactment service will allocate the activity to a suitable workflow engine by placing the activity in the jobs queue of that engine. The activity is now in the *Queued* state. Once the engine becomes free and starts executing the activity, the state of the activity becomes *Active*. The *Completed* state means that the activity has finished execution (either successfully or not). Failure to execute an activity will result in termination of the process instance execution.

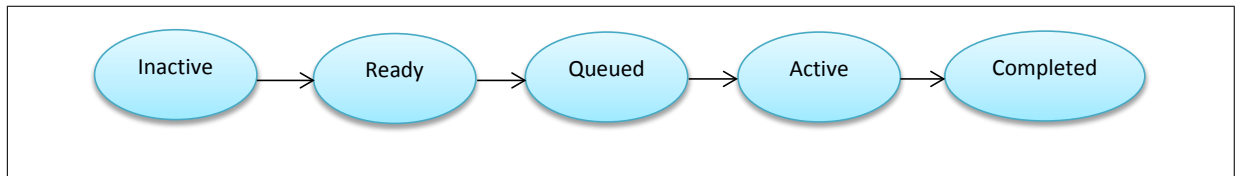


Figure 2.6: Activities life-cycle

2.5.5 Artefacts life-cycle

Artefacts are consumed/produced by activities while a workflow instance is being executed. In addition, artefacts can be created offline by actors and provided as input for activities. In some cases, artefacts will be edited by actors or activities during workflow instance execution. The new modifications are stored as a new version of the same artefact. Figure 2.7 illustrates this cycle.

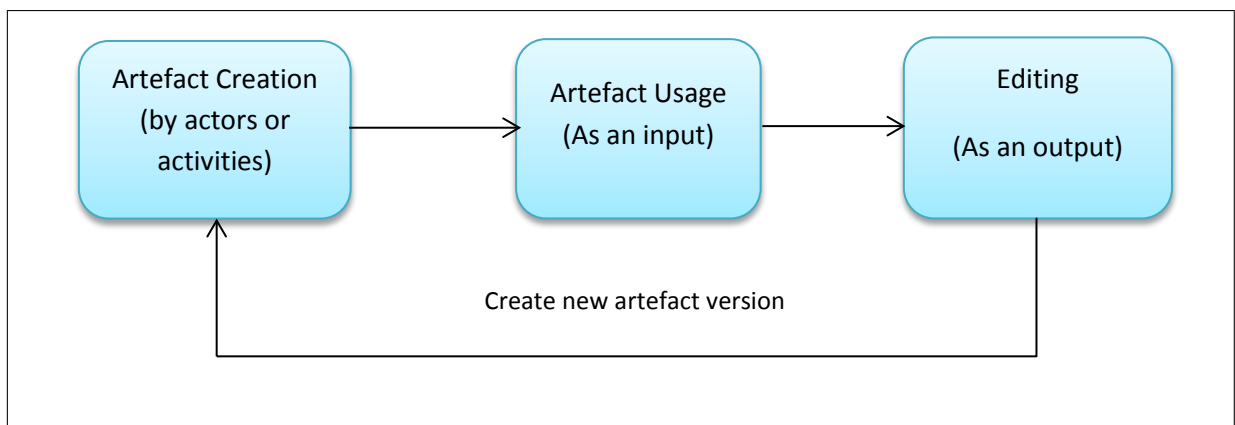


Figure 2.7: Artefacts life-cycle

2.6 Proof of Concept

As an initial evaluation of the SDaaS reference architecture, we instantiate a proof-of-concept prototype implementation which demonstrates the feasibility of the SDaaS vision. In this section, we describe this prototype implementation.

The prototype demonstrates some of the core features of the SDaaS architecture but does not implement all components. We list the supported features here and then in the following subsections, we describe the technical implementation, deployment and the tools we integrated into the prototype.

The instantiated prototype supports the following features:

- Interpreting software process models described in an XML format which will be covered in Chapter 3.
- Schedule software process activities for execution in the cloud.
- Execute these activities on a set of distributed workflow engines in the cloud.
- Manage the execution and capture provenance data.
- Store and manage software artefacts and their meta-data.

The prototype scheduler is implemented to apply the Proportional Adaptive Task Schedule algorithm (see Chapter 4) to dynamically allocate activities to workflow engines matching their requirements. The workflow engines are separate applications which interact with the enactment service through asynchronous communication channels and the REST API. The prototype handles simple control flows such as forks and joins. However, it does not support loops, decision points and interactive activities. Furthermore, the prototype does not provide features for building software process models nor for providing SLA monitoring, consistency checking or external workflow collaboration. These features are left for future studies.

2.6.1 Implementation & deployment

The prototype is implemented as two Java enterprise applications, a message-oriented middleware and a document-based NoSql database. The NoSql database provides scalability and supports storing artefacts as documents. The two applications are the enactment service and the workflow engine which are both implemented as a webservice using Spring 3.0 framework ¹.

In order to decouple the enactment service from the workflow engines, asynchronous communication between them is achieved through message oriented middleware. The enactment service pushes jobs to workflow engines by placing the job into their designated jobs queue. The workflow engines place progress updates into the enactment service responses queue. Figure 2.8 illustrates the communication model (using ActiveMQ ² as a messaging middleware).

¹<https://projects.spring.io/spring-framework/>

²<http://activemq.apache.org/>

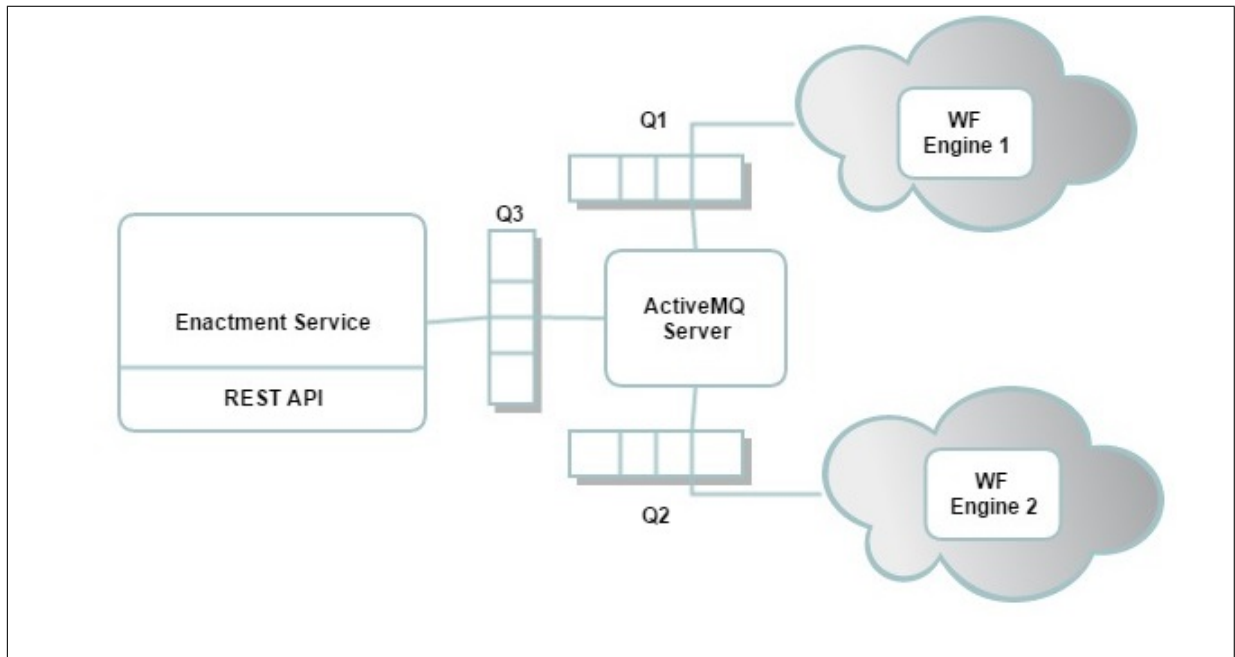


Figure 2.8: Message oriented communication

The NoSql database used is MongoDB³. Artefacts (including process models) and their meta-data are stored as documents in MongoDB. This combination of technologies allows the prototype to scale and provides decoupling between the different components. The prototype is deployed on Amazon AWS public cloud as a representative of commercial public clouds. However, the prototype can be deployed on any IaaS provider(s).

2.6.2 Migrated tools

As part of this initial evaluation, we wanted to execute some small processes. In order to model and execute such processes, we needed activities to be created and integrated in the SDaaS prototype. We have taken three existing tools and wrapped them as SDaaS activities and integrated them in the prototype. This relates to requirement *R2* in Section 2.3; the availability of tools in real time. These tools represent examples of: computationally intensive tools (Spin), distributed tools (DiVinE) and tools extracted from other environments/tool-sets (Concerto-FLA). Although these tools do not represent all types of tools that can be supported, they represent tools that can benefit from the computational power and scalability of the cloud.

³<https://www.mongodb.com/>

2.6.2.1 Spin

Model checking is a computationally intensive task where models describing systems behaviours are checked to verify if certain properties hold or not. This task often requires traversing large state spaces. This demands vast computational resources and often faces the “state explosion” problem. Spin [59] is a model checker based on reachability analysis. It focuses on concurrent asynchronous systems (software rather than hardware). Spin accepts models written in Promela [58] which describe the behaviour of the system. In addition, it accepts the correctness claims that need to be proved and verified for the model. The correctness claims are expressed in Linear Temporal Logic (LTL). They are used to formalise the system’s erroneous behaviours and they are transformed into Buchi automaton [106]. The correctness properties are categorised into two categories: safety properties and liveness. The former means that nothing bad happens while the later means that something good eventually happens.

Spin works as follows: first, model a concurrent system in Promela and parse it. Then, correctness claims are expressed. Interactive simulation is run to ensure that the model describes the system’s behaviour correctly. Finally, an on-the-fly verifier program is generated to verify the model.

Spin has been integrated into the eScience platform as described in Chapter 1. We also, wrapped it as an activity and integrated it in the SDaaS prototype implementation. The activity takes as an input the Promela model and performs the model checking then generates a file which contains the Spin textual output.

2.6.2.2 DiVinE

Unlike Spin, DiVinE [23] is a distributed model checker which works on a network of single/multi-core machines. Distributed model checkers require heavy synchronisation between all participating machines. This synchronisation is needed to split the state space between the machines and ensure that it has been fully explored. DiVinE accepts input models in multiple formats such as LLVM and MurPHI.

Similar to what we did with Spin, we wrapped the DiVinE tool as an activity and integrated it in the SDaaS prototype implementation. This activity takes as an input the model to be checked as well as a list of parameters specifying the number and type of

machines to be used for the distributed model checking task. Those parameters define how and when to scale the number of participating machines up.

2.6.2.3 Concerto-FLA

Failure Propagation and Transformation Calculus (FPTC) is a failure logic analysis allowing for the calculation of the system level failure behaviour based on the failure behaviour of the individual components. The propagation of failures from the inputs to the outputs of a component is captured via FPTC rules. More details about FPTC is provided in Chapter 5.

We extracted Concerto-FLA [55] (the FPTC analysis component from the CONCERTO project tool-set⁴) as a standalone application. This application is then integrated as an activity within the SDaaS prototype implementation. The CONCERTO tool-set allows: creating UML-based architectural models of the system and performing FPTC analysis (using Concerto-FLA) including back-propagation of the results on the models. The architectural model is transformed to the *flamm* format (an XML-like format) on which the FPTC analysis takes place. This activity has been used in the case study presented in Chapter 5.

2.7 Discussion

In this section, we discuss how the SDaaS reference architecture -proposed in Section 2.4- can potentially address the requirements defined in Section 2.3. It is worth noting, that complete fulfilment of the requirements is highly dependent on the domains and scenarios where the SDaaS architecture is used.

R1: Awareness and synchronisation support and *R3: Organisational policy convergence* can be met by using software process models. Process models are used for communication, documentation and execution of processes. The process models are used as a base for spreading awareness and monitoring the progress. Additionally, these process models reflect the organisational policies and standards. For *R2: Availability of tools in real time*, the *Artefact Repository* stores activities which are retrieved by the *Execution Manager* when needed for process execution. Compatible external tools are called through the

⁴<http://www.concerto-project.org/>

External Tools module. *R4: Capturing process and provenance data* can be met by the *Execution Manager* module which collects provenance data about the process model execution and the generated artefacts. *R5: Accessible artefacts* can be met through both the *Artefact Repository* and the *REST API*. The API provides endpoints to access and manipulate artefacts from the artefact repository. *R6: Governance and inter-organisation collaboration* can be met by the *External workflow collaboration* module which allows the SDaaS architecture to interact with other platforms and pass certain artefacts to trigger partner's process execution. The *SLA Monitor* module monitors if all partners are complying with the SLA while executing their parts of the project.

R7: Privacy and legal compliance can be met through the fine-grained configuration of process models execution. EXE-SPEM (as we will explain in Chapter 3) allows process activities to be configured differently for execution as part of the process modelling. The configurations allow an activity to be executed only in a private cloud for instance. Combined with the fact that the SDaaS architecture can be deployed in a public, private or hybrid cloud, the fine-grained configurable models meet the privacy and legal compliance needs.

Since the SDaaS architecture is designed to support multiple teams and multiple processes belonging to multiple software projects, then *R8: Multi-tenancy* can be met. The *Model Authoring* and *Access & Sync. Service* provide support for the tenants to create, access, edit and execute process models and artefacts.

R9: Scalability can be met through the scalability of the workflow engines which host the execution of individual process activities. The workflow engines are scaled up and down on demand. As we will see in Chapter 4, the *Scheduler* decides to scale workflow engines up and down on periodic bases to reduce the execution cost and improve workflow engines utilisation. Additionally, the *asynchronous communication* between the enactment service and the different workflow engines (using message queues) also supports the scalability of the entire SDaaS architecture. This is because neither the enactment service nor the workflow engines have to interact with the message queue at the same time.

Since the SDaaS reference architecture is not designed to be domain specific, we cannot consider the set of requirements we use to be complete. Meeting those requirements is highly dependent on the chosen tools, deployment environments and domains where

the SDaaS architecture will be used. Some of the requirements can trade-off against each other. For example, the choice of tools which are not designed to utilise scalable resource would trade-off against the scalability requirement. Therefore, the suitability of the SDaaS reference architecture depend on the scenario and domain where it will be used.

2.8 Summary

In this chapter, we have proposed the Software Development as a Service (SDaaS) reference architecture and its components. We based the reference architecture on a list of requirements and discussed how it meets them. Additionally, we provided an initial evaluation through instantiating a prototype of the architecture. The prototype and the tools we integrated into it demonstrate the feasibility of the SDaaS vision. The next two chapters focus on specific aspects of the SDaaS architecture; Chapter 3 covers the modelling of cloud-based executable software processes and Chapter 4 discusses the execution scheduling of such processes in the cloud. Chapter 5 evaluates the SDaaS architecture using a case study which uses its prototype for enacting safety-related processes.

3

MODELLING SOFTWARE PROCESSES FOR CLOUD-BASED EXECUTION USING EXE-SPEM

Contents

3.1 Introduction	39
3.2 Background	40
3.2.1 Software process modelling	40
3.2.2 Software process modelling standards	42
3.3 Requirements for Cloud-Based Executable Software Process Models . .	45
3.4 EXE-SPEM	48
3.5 Model to Text Transformation	51
3.6 Sample Process	53
3.7 Discussion	54
3.8 Summary	55

An earlier version of this chapter is published in: S Alajrami, B Gallina, A Romanovsky, **EXE-SPEM: Towards Cloud-based Executable Software Process Models**. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development MODELWARD'16. Rome, Italy. pp. 517-526, 2016.

3.1 Introduction

Modelling software processes has several benefits. Curtis et al. [39] list some of these benefits; for example, facilitating understanding and communication and provision of automated execution support. Modelling software processes either target automation (focusing on models for machines) or process improvement (focusing on models for humans which are used for process evaluation, improvement, management, etc.) [88]. Both types of models are useful during certain phases of the development life-cycle. Generally, the longer the model is in use, the more value is gained from it. If software process models are executable, they will be in use throughout the development process phases. For instance, an executable software process model will be used to manage, monitor and execute the process through all life-cycle phases rather than being only used in the design phase for documentation/communication. Thus, the overhead cost of modelling will be justified. It is worth noting that we use the term execution/enactment in this thesis differently from other studies. While some studies (e.g., [45, 93]) refer to process monitoring and tracking as enactment, we define process enactment as: the act of controlled execution of the process activities -either automatically through software tools or by actors using software tools- and producing the expected artefacts.

In this chapter, we focus on the *Process Modelling* part of the SDaaS reference architecture (see Figure 3.1). We propose EXE-SPEM which is an extension of the OMG Software and Systems Process Engineering Meta-model (SPEM2.0) standard. EXE-SPEM enables modelling of cloud-based executable software process models.

We explore the state-of-the-art software process modelling languages and analyse their suitability for enacting cloud-based software processes. As we explain in Section 3.2, none of the existing software process modelling languages have enough support for model execution and/or cloud-based execution. However, SPEM2.0 has the capability of modelling most elements of software processes, therefore, we extend its meta-model and propose EXE-SPEM in Section 3.4. We then provide an XML schema for a machine executable counterpart of EXE-SPEM models in Section 3.5. As an example, we model the Facebook continuous delivery process from Chapter 1 using EXE-SPEM in Section 3.6.

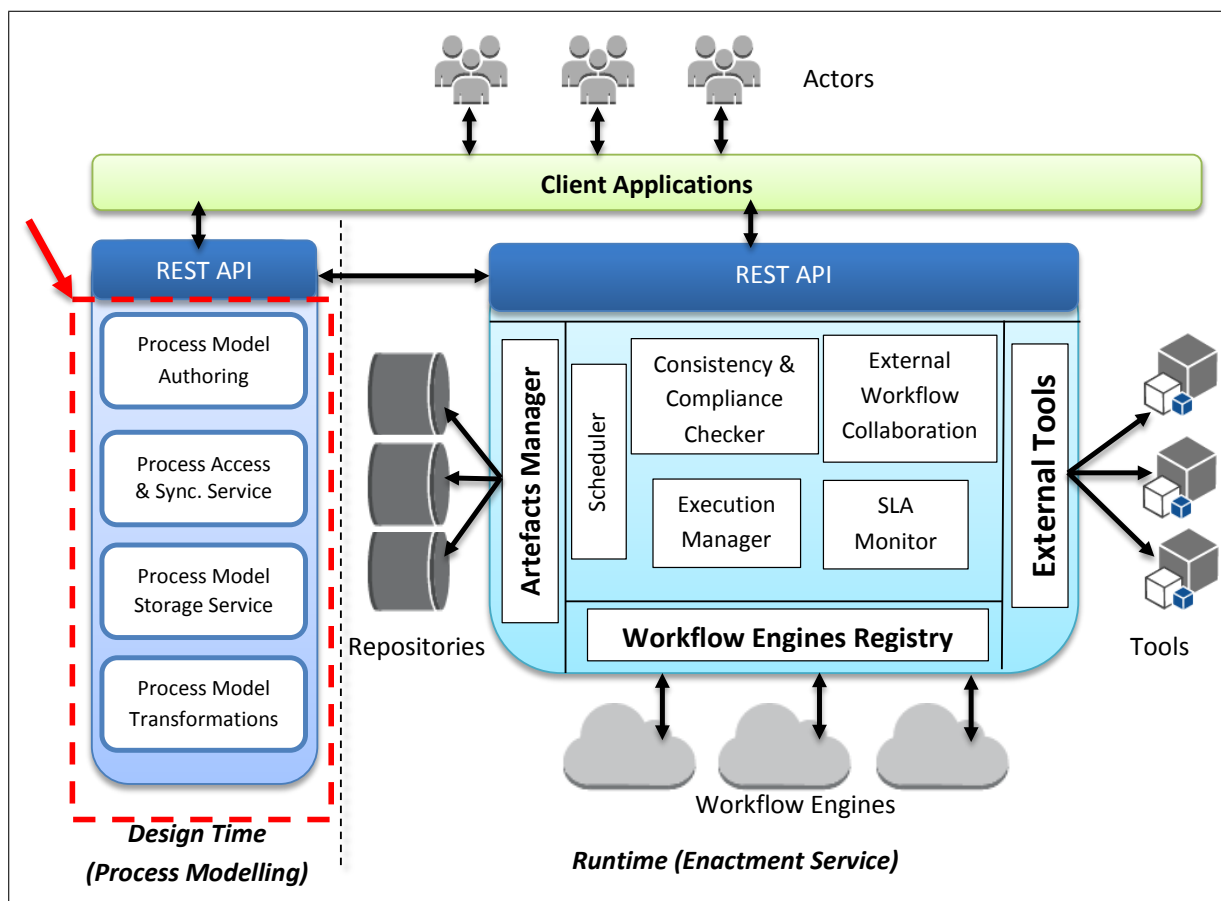


Figure 3.1: The process modelling components in the SDaaS reference architecture

3.2 Background

In this section, we present background information on different software process modelling approaches and standards and examine their suitability for modelling cloud-based executable software processes.

3.2.1 Software process modelling

The evolution of software development paradigms has focused on increasing the level of abstraction and automation in software development to enable developers to focus on the core business logic. Models (and Model-Driven Engineering (MDE)) have been the main means of achieving such abstraction and automation.

Software process models have two aspects: the production aspect (focusing on conceiving software products) and the management aspect (focusing on planning and managing the resources needed for the production) [12]. Therefore, different models can represent different points of view. Acuña et al. [12] list the different process el-

elements that can be contained in a process model such as agents, activities, artefacts, roles and events.

Modelling software processes has been investigated since late 80s. There are many motivations which led these investigations, including:

- Improving the understanding for different perspectives by visualising the relevant components for each perspective.
- Facilitating communication among team members.
- Supporting project management through reasoning in order to improve the process.
- Partially automating processes through Model-Driven Engineering (e.g., repetitive and non-interactive tasks).

Several approaches to software process modelling have been introduced over time, they are categorised into four categories [25]:

1. Rules based (e.g., MARVEL [74])
2. Petri net based (e.g., SPADE [22])
3. Programming languages based (e.g., SPELL [37])
4. UML based (e.g., SPEM2.0 [5])

The first three did not receive industrial take up due to their complexity and inflexibility [64]. The UML approach is based on utilising the wide adoption and acceptance of Unified Modelling Language (UML) for modelling software processes. Several implementations of this approach have been proposed each with different strengths and weaknesses. The authors of [25], compare six UML-based modelling approaches based on a set of software process modelling requirements. They also admit that executability and formality are major weaknesses of using UML for software process modelling.

Today, there are standards for modelling software processes which are endorsed by standardisation bodies and academic communities. The next section discusses a sample of these standards.

3.2.2 *Software process modelling standards*

In this subsection, we provide a brief overview of three software process modelling standards (SPEM2.0, Essence and ISO 24744) and we evaluate them in order to decide which one to use/extend.

3.2.2.1 SPEM2.0

SPEM2.0 [5] was developed by the Object Management Group (OMG) for defining software and system development processes and their components. With the aim of accommodating large range of development methods and processes, SPEM2.0 was designed to be generic without adding domain-specific elements to its core structure. SPEM2.0 is defined as an MOF-based meta-model and a UML 2 profile [5]. It is based on the concept of interaction between *Roles* that perform *Activities* which consume (and produce) *Work Products* [35]. SPEM2.0 is structured into seven meta-model packages which contain its modelling elements.

One of the problems with SPEM2.0 is its lack of explicit enactment support. In Section 16 of the SPEM2.0 specification [5], it is stated that there are two common ways for enacting SPEM2.0 process:

- Mapping the process model into project plans and enacting them using project planning tools.
- Mapping the process model to a business flow or execution language then enacting it in a workflow engine.

As a result of the lack of enactment support, several researchers have proposed different approaches and extensions to support process enactment. In [117], the authors propose mapping rules to map SPEM2.0 models into XML Process Description Language (XPDL) which then can be enacted in XPDL-based engines. In [93], authors propose xSPIDER_ML (a software process enactment language based on SPEM 2.0 concepts). Although xSPIDER_ML is supported with a modelling tool and an enactment environment, the notion of enactment is limited to process monitoring since developers are supposed to perform their tasks off-line and report their progress to the enactment environment. The authors in [45] introduce eSPEM which is a SPEM

extension to allow describing fine-grained behaviour models that facilitate process enactment. They implement a distributed process execution environment [46] based on the Foundational subset for Executable UML Models (FUML¹) standard with emphasis on supporting the ability to share process state on different nodes, suspend and resume process execution, interact with humans, and adapt to different organisations. However, the notion of process enactment in that execution environment also assumes that developers carry out their tasks outside the execution environment and return control back to it once they finish. Additionally, there are SPEM2.0 extensions which address specific domains' needs. For instance, S-TunExSPEM [57] allows modelling and simulation of safety-oriented processes based on safety standards (e.g., DO-178B). To support executability, the authors define mapping rules between S-TunExSPEM and XPDL2.2 [111].

In general, we found that all these SPEM2.0 extensions have one or more of the following weaknesses:

- These extensions do not have any available tool support.
- Their notion of enactment is limited to monitoring the process while the process itself is performed completely outside the enactment environment.
- They do not have explicit support for cloud-based enactment.

3.2.2.2 ESSENCE

Essence - Kernel and Language for Software Engineering Methods [10] was initiated by the Software Engineering Method and Theory (SEMAT) initiative as a response to the Request For Proposal "A Foundation for the Agile Creation and Enactment of Software Engineering Methods" from OMG. Essence provides process elements (some of which are similar to SPEM2.0 elements). The main language concepts are: *kernel*, *practises* and *methods*.

The Kernel is "a light-weight set of definitions that captures the essence of effective, scalable software engineering in a practice independent way" [10]. It is organised into three areas of concern; customer (the users), solution (the system) and endeavour (the team and process). Each area contains a set of:

¹<http://www.omg.org/spec/FUML/>

- *Alphas*: things to manage, use and produce by the team (e.g., *Requirements*).
- *Activity Spaces*: things to do when developing and maintaining the system (e.g., *Understand Stakeholder Needs*).
- *Competencies*: the capabilities required to carry out the work (e.g., *Leadership*).

A practice is “a repeatable approach to doing something with a specific objective in mind” [10]. It describes how to handle certain aspects of the software development endeavour (e.g., Scrum for agile project management). Method is “the composition of a Kernel and a set of Practises to fulfil a specific purpose” [10]. Essence is useful for instructing and guiding development teams. For that, Essence embeds guidance in all elements and as a result, the process model contains large amount of natural language description of such guidance. Elvesæter et al. [47] compared SPEM2.0 and Essence to evaluate their applicability for agile processes and enactment support. They conclude that although there are similarities in the process authoring capabilities, Essence has better support for enactment compared to SPEM2.0 which does not have any enactment support. However, they refer to support for monitoring and tracking the process which is (as discussed in Section 3.1) not the type of enactment we are after.

3.2.2.3 ISO 24744

The Software Engineering Meta-model for Development Methodologies (SEMDM)[4] is an industrial ISO standard which is based on concepts adopted from the OPEN process framework [51] and from method engineering concepts. SEMDM aims to define methodologies in information-based domains which rely on information management and processing [4]. It uses a different conceptual approach to SPEM2.0 and Essence and its process meta-model is based on *power type* pattern and on a set of so-called *Clabject* constructs [104]. In its current state, SEMDM is a documentation of the standard with no reference implementation available and very little academic attention [79].

3.2.2.4 Choosing SPEM2.0 for software process modelling

After reviewing the standards above, we can see that each standard had its strengths and weaknesses. All the reviewed standards do not support modelling cloud-based enactment requirements and do not have any native means of supporting enactment

which executes the process and not just monitors it. However, the standards come with an out-of-the-box set of elements that can be used to model software processes. Therefore, the first decision we make here is to adapt one of the standards and enrich it with any missing elements rather than reinventing the wheel.

The second decision is which standard do we adapt? By looking at Table 3.1, we can see that both SPEM2.0 and Essence have similarities in process authoring elements while ISO 24744 uses a totally different approach. The key selling point of ISO 24744 is that it focuses on the product rather than the process and promoting just-in-time enactment of processes rather than sticking with rigid one-off processes. However, ISO 24744 has not received much academic attention nor industrial adoption. It did not even have any reference implementation of the standard [104] and we are not aware of any tool-support available for it. Therefore, we eliminate it as an option.

We choose SPEM2.0 over Essence to model software processes for the SDaaS architecture since SPEM2.0 is more mature and has received more academic attention. While Essence is perceived to be better for agile processes, we recommend using SPEM2.0 elements to model smaller and more fine grained processes which allows for just-in-time enactment. In addition, Essence enactment refers to process monitoring while SPEM2.0 recommends mapping SPEM2.0 models into workflow engines formats for enactment (which is what we do in the SDaaS architecture).

3.3 Requirements for Cloud-Based Executable Software Process Models

In this section, we define what information a cloud-based software process model should contain to enable cloud-based model execution in the SDaaS reference architecture. These requirements are identified by inspecting the basic information needed to allow a software process model to be executed in the cloud and benefit from its scalability. These needs were discovered through the attempt to use an existing workflow management system (eSC [66]) as we described in Section 1.3.3. However, these requirements are not complete and domain-specific processes may require more information to be incorporated in software process models. The requirements are:

²https://eclipse.org/epf/downloads/tool/tool_downloads.php

³<https://www.ivarjacobson.com/esswork-practice-workbench>

Table 3.1: Comparing the three software process modelling standards

Criteria	SPEM2.0	Essence	ISO 24744
Maturity	Mature with larger community and attention.	Relatively new with little attention so far.	Very little academic attention and case studies.
Authoring Elements	Has some elements which Essence does not have. e.g., Roles. Guidance is a specific element. Uses less natural language.	Has alternative ways of expressing elements that SPEM2.0 has. e.g., Role. Guidance is embedded in all elements. Uses more natural language.	Uses different concepts from the OPEN process and method engineering.
Application	Applicable for software & system processes in general. Supports defining breakdown structures which allows modelling different processes.	Suitable for agile processes. Does not support breakdown structures as Agile methods downplay them and replaces them with sprints (increments).	Suitable for system engineering with focus on software engineering.
Enactment	No explicit enactment support. Recommend mapping to project management or workflow tools.	Supports enactment in the form of process monitoring and tracking.	Focuses on the product and advocates just-in-time enactment rather than the one-off variant.
Cloud-related & modelling	Not supported.	Not supported.	Not supported.
Tool Support	EPF Composer ² .	EssWork Practice Workbench (EWPW) ³ .	N/A.

- **R1- Allow defining the required cloud resources for an activity.**

Software process activities are diverse and they use different tooling support. While some activities in a process might be as simple as editing a textual file, other activities could involve more complex computational tasks (e.g., the distributed model checker DiVinE which we came across in Chapter 1). Such computationally intensive tasks need to be allocated the appropriate computing resources. With the elasticity of cloud computing, it is possible to allocate an initial set of resources and scale it up and down as needed. A cloud-based software process model needs to capture the initial set of resources needed for each activity to start execution. It also needs to capture the resource scaling mechanism if needed. To cater for different activities' needs, the model should allow having different execution configurations for each activity.

- **R2- Allow defining security and privacy measures.**

Security and privacy are critical concerns when using cloud computing [114]. Many enterprises and officials are sceptical about using public clouds based on their fear of data loss or breaches. Although cloud computing relieves enterprises from infrastructure management and maintenance, this comes with the disadvantage of cloud's opacity. Users do not know where their data is actually located and which other users may have access to it. Private clouds came to address those concerns by giving full control over the infrastructure to the user. In a software process model, some activities may use confidential or sensitive artefacts. Therefore, process authors should be able to define whether an activity (and its artefacts) should be executed in a private cloud (for security and privacy reasons) or in a public cloud.

- **R3- Define basic human-machine run-time interactions.**

Software processes are very complex and involve many stakeholders (e.g., designers, developers, project managers, business analysts, customers, etc.). While in some cases the process activities can be repetitive and automated (with no or little human interactions), many activities would require human interactions during the process execution. We envision to support two types of basic human-machine interactions:

- Decision making: where a human would guide the executing process at run-time by specifying a particular branch the execution should follow, or by deciding to repeat a particular activity with different settings. This kind of interaction should be defined in the process model in order to be supported at run-time.
 - Parameter passing: in some cases, it might be difficult to set some execution parameters for an activity at the modelling stage. In such cases, a simple interaction is needed to pass those parameters at run-time. This allows activities to have a simple interaction with users in the form of questions (asking for parameters) and answers (passing the parameters by users).
- **R4- Allow defining control-flow semantics.**

Software processes are control-flow processes. Software process models need the flexibility of expressing control flow semantics such as: loops, forks and joins.
 - **R5- Allow defining the required tool support.**

Activities in software processes are usually supported by some tools. In this context, activities are used as wrappers for tools and the execution of software process models means orchestrating these tools in a workflow style. Therefore, the model need to incorporate the tool (activity) details such as: version and compatible inputs and outputs.

3.4 EXE-SPEM

As described in Section 3.2, the chosen standard to use for modelling cloud-based executable processes is SPEM2.0 and as detailed in Section 3.2.2.1, SPEM2.0 does not have explicit support for process execution. In addition, the existing SPEM2.0 extensions do not satisfy the requirements for cloud-based executable software processes as listed in the previous section. In this section, we extend the SPEM2.0 meta-model to address these requirements. We call the extended version EXE-SPEM (Executable SPEM).

Out of the seven SPEM2.0 meta-model packages, the *Process Structure* meta-model contains the structural elements for process definition. EXE-SPEM extends the *Process Structure* meta-model with two new meta-classes, one enumeration and adds attributes to existing meta-classes. Figure 3.2 illustrates the extended meta-model where

meta-classes with dark grey background are new and meta-classes with light grey background have new attributes.

The extension is summarised in the following points:

- The *CloudPrivacyKind* enumeration is added to define types of cloud deployment where an activity will be executed. It is used as an attribute in the *Activity* meta-class.
- The *Activity* meta-class is extended with attributes that will be used to guide execution in the cloud. The added attributes specify the version of the activity (the supporting tool) to be used, the type of cloud deployment (private or public) and the type and number of machines and a timeout for executing the activity in the cloud. Additionally, a priority flag specifies if an activity must be executed immediately regardless of the cost or not. This meets the requirements R1 and R5 from Section 3.3. The use of *CloudPrivacyKind* here satisfies requirement R2. Additional optional attributes are added for safety-related processes like the one used in the case study presented in Chapter 5. These are:
 - *Standard* which denotes the particular standard recommending the use of this activity.
 - *Guidance* which is the guidance used for guiding the use of activity.
 - *Tool Qualification* which refers to any qualification the tool used to support the activity has got.
- Two subtypes of *Activity* are introduced to provide control flow semantics:
 - The *Control Point* provides the semantics of control flow in the process model. Control points in the process model give the user executing the process the ability to decide which branch the execution should follow next. A branch can be: a loop (referring to the same activity), a fork or a join. The control point interaction is simply done by providing options (pre-defined in the model) and asking the user to make a decision on which option to follow. This meets requirement R4 and the decision making interaction partially meet requirement R3.

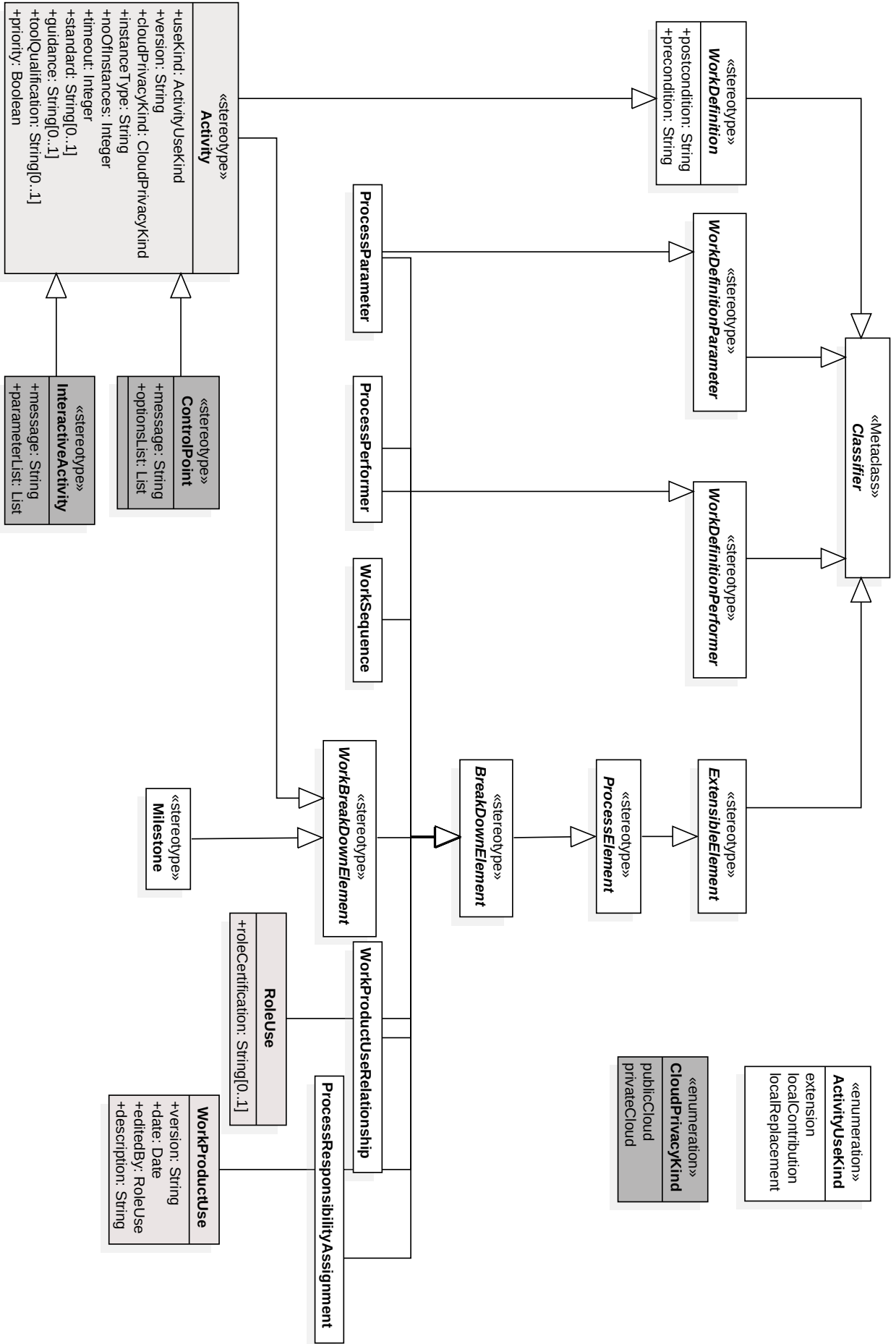







Figure 3.2: The meta-model of the extended SPEM 2.0 process structure

Table 3.2: Graphical icons of EXE-SPEM elements

Element	Icon	Element	Icon
Process		Activity	
Interactive Activity		Control Activity	
Work Product Use			

- *Interactive Activity* which can be used to model an activity that involves simple interactions with stakeholders. This meets the parameter passing type of interactions in requirement R3.
- The *RoleUse* meta-class is extended with an attribute providing information about the certification held by this role.
- The *WorkProductUse* meta-class is extended with attributes providing information about the work-product (the artefact) including: the version, the date, the description and the last role who edited it.

Icons for the EXE-SPEM elements are provided in Table 3.2. It is worth noting that EXE-SPEM reuses some of the SPEM2.0 elements (Role Use, Guidance, Process Parameter and Work Sequence) with the same icons.

3.5 Model to Text Transformation

In order to execute EXE-SPEM models, we map them to a machine-executable XML format following the XML schema in Appendix A. The meta-model of this format is described in Figure 3.3. The XML schema captures a process consisting of the following elements:

- **Process:** this is the software development cycle. A process is usually created by an actor but might be executed by multiple actors.
- **Actor:** a person who is involved in the process such as: process managers, software engineers, testers, etc. A process will involve one or more actors. Although a team of actors might collaborate off-line on performing an activity, the activity will be assigned to a single actor who takes the responsibility for this activity.

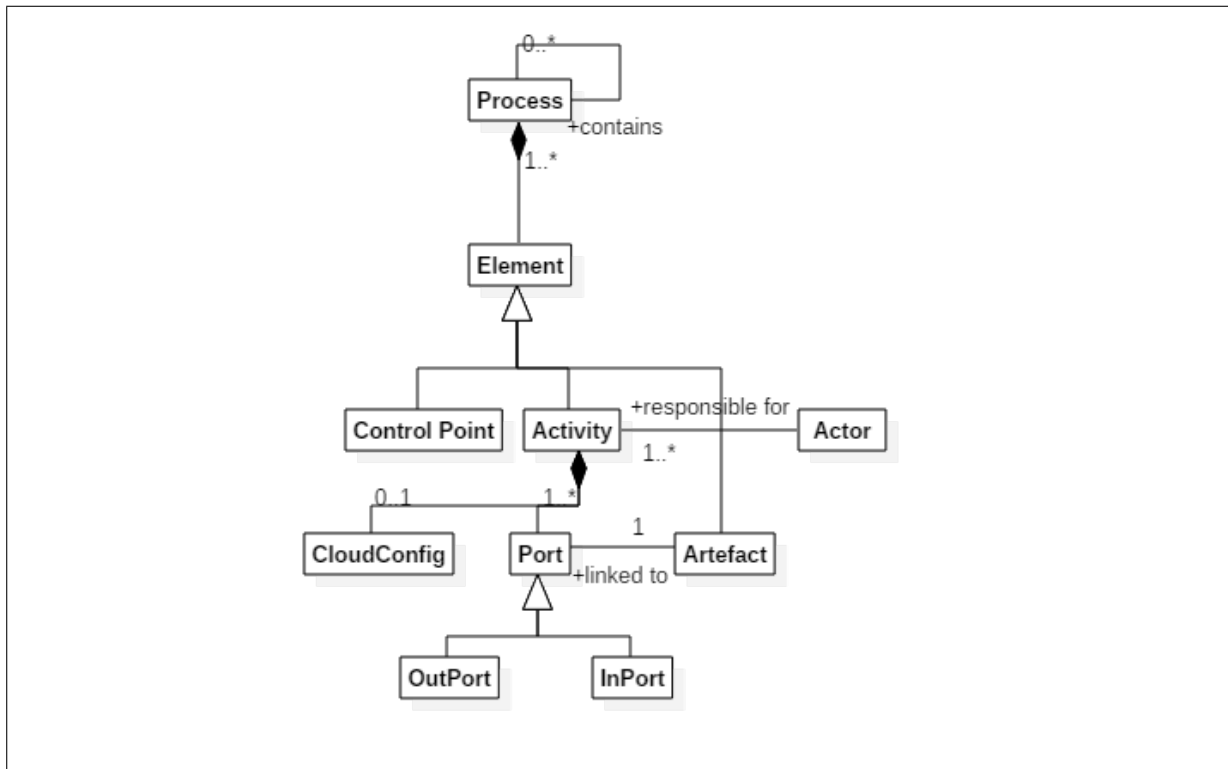


Figure 3.3: The meta-model of the XML format

- **Artefacts:** items produced or needed by the activities of the software development process (e.g., code, executables, models, documents, etc.).
- **Activities:** Activities represent the smallest unit of execution. They represent the different steps in a software process. Those steps usually involve the use of tools and/or actor interaction to be completed. Activities can be:
 - *Concrete activities:* are executable blocks of code. This type of activities is the tool support that is used for process execution. For instance, a verification activity will be supported by a verification tool (e.g., a model checker) which will be executed.
 - *Control points:* a type of activities which allows actors to guide the execution of the process in one of multiple pre-defined directions. This allows for supporting loops, if conditions, and forks.
- **Cloud configuration:** represents cloud-related configurations such as: cloud deployment type, machine type, machine image and number of machines to be used.
- **Ports:** Each activity can have zero or more input ports and zero or more output

Table 3.3: Mapping rules between EXE-SPEM and our XML notation

EXE-SPEM cloud-specific process	XML Process
Process	Process
Phase	(Sub) Process
Activity	Activity
Control Point	Control Point
Interactive Activity	Interactive Activity
Activity (execution -related) attributes	Cloud Configuration (of an activity)
Task	Activity description attribute
Work Product Use	Artefact
Role Use*	Activity actor attribute
Guidance*	Activity description attribute
Process Parameter*	Port
Work Sequence*	Port attributes

ports. Ports provide the means to connect activities and direct the process execution flow. They define both the consumed and produced artefacts/parameters by an activity. In addition, input ports act as preconditions that need to be satisfied so that the activity can start executing.

Table 3.3 shows the rules to map an EXE-SPEM model into the XML format described above. The mapping include some SPEM2.0 elements which are reused in EXE-SPEM. These elements are denoted with *. Algorithms to automate this mapping are not implemented yet and are considered future work.

3.6 Sample Process

After introducing EXE-SPEM and the rules for mapping EXE-SPEM models into a machine-executable XML format, in this section, we model the Facebook continuous delivery process taken from [49]. The process is used by Facebook to continuously implement, test, deploy and release new features to users. It involves different stakeholders and few control points where decisions are made about releasing the features or not and to which users.

Figure 3.4 shows the process modelled in EXE-SPEM. For clarity, not all artefacts have been labelled but all unlabelled ones refer to the *source code* of the new feature as it is

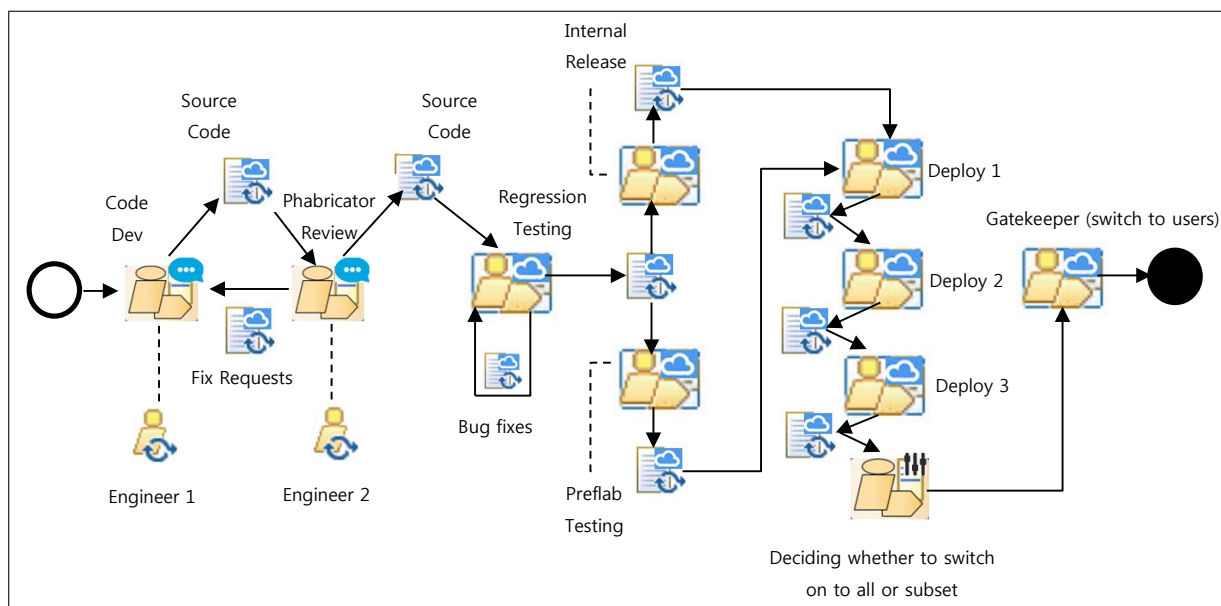


Figure 3.4: Facebook's continuous delivery process model in EXE-SPEM

being evolved through the process. In addition, all the activities will be carried out by engineers from the team working on the feature, however, to simplify the figure, we only included two actors. We can see that the *Phabricator Review* is an interactive activity where *Engineer 2* checks the code from *Engineer 1* using the Phabricator tool and requests fixes for any discovered bugs. The *Regression Testing* activity can take several iterations as bugs are being discovered and fixed by the engineers. The *Deploy 1,2,3* activities represent gradual deployment of the new feature. Finally, there is a control activity deciding whether to switch the new feature to all users or to a subset of them before this is actually performed by the *Gatekeeper* activity.

This process model is mapped to an XML model following the mapping rules in Table 3.3. The full XML process model can be found in Appendix B.

In contrast to SPEM2.0, using EXE-SPEM in the above model has allowed to model interaction and control flow semantics. In addition, the textual model (XML) contains configurations that will be used for enactment by the enactment service as we already explained in Chapter 2. Another process example modelled in EXE-SPEM can be found in the case study presented in Chapter 5.

3.7 Discussion

In this section, we discuss how the proposed software process modelling language; EXE-SPEM meets the requirements for modelling cloud-based executable software

process models to be executed in the SDaaS architecture.

EXE-SPEM models are built using an extended set of SPEM2.0 elements. The extension of SPEM2.0 is done by extending its meta-model and then mapping the process model to an execution language which can be executed in a workflow engine. This approach is suggested in the SPEM2.0 standard [5]. The visual EXE-SPEM models are mapped to an XML format which can then be parsed and executed in the SDaaS architecture. Similar approach has been used for other SPEM2.0 extensions (e.g., [57]).

The requirements we described in Section 3.3 are met by the extension made to the SPEM2.0 meta-model as shown in Figure 3.2. *R1* is met by the attributes added to the *Activity* meta-class specifying the required computational resources. *R2* is met using the *CloudPrivacyKind* meta-class which specifies the privacy option for the workflow engine that will execute a certain activity. Both *Interactive Activity* and *Control Point Activity* meta-classes meet requirement *R3*. The *Control Point Activity* meta-class also meets requirement *R4*. Finally, *R5* is met by the attributes defined in the *Activity* meta-class to specify the required tool support for an activity.

3.8 Summary

In this chapter, we focused on the process modelling part of the architecture presented in Chapter 2. We introduced EXE-SPEM, the SPEM2.0 extension which supports modelling cloud-based executable software process models. We analysed the suitability of three modelling standards for cloud-based executable software process modelling. We chose SPEM2.0 because of its maturity and extensibility.

We detailed the extension to the SPEM2.0 meta-model and how EXE-SPEM can be mapped into an executable XML format. Now that we can model cloud-based executable software processes, the next step is to start enacting the process in the cloud. The next chapter focuses on how to schedule process activities on the right workflow engines in a cost-efficient way.

4

COST-EFFICIENT SCHEDULING OF SOFTWARE PROCESSES EXECUTION IN THE CLOUD

Contents

4.1	Introduction	57
4.2	Background	57
4.2.1	Workflow scheduling	58
4.2.2	Workflow scheduling algorithms	61
4.3	Scheduling SDaaS Software Workflows in the Cloud	65
4.3.1	Assumptions	65
4.3.2	Objectives	66
4.3.3	Motivation	66
4.3.4	Problem definition & assumptions	67
4.3.5	Scheduling requirements	70
4.3.6	Cost factors	71
4.3.7	Scheduling algorithms	72
4.4	Evaluation	78
4.4.1	The request generator	78
4.4.2	The simulation scheduler	80
4.4.3	Workflow engines	80
4.4.4	Performing the simulation	81
4.4.5	Simulation results	81
4.5	Summary	86

4.1 Introduction

Executing workflows in the cloud harnesses the cloud economies of scale. However, despite the illusion that the cloud offers an unlimited pool of computational resources, these resources come at a cost. While computational resources might be plenty, monetary resources are always limited. Therefore, in this chapter we investigate the software workflows execution scheduling and how we can reduce the cost of execution without causing significant execution delays.

In a setting where multiple software development workflows (and their activities) compete for shared computational resources (workflow engines), scheduling workflow execution becomes important. Workflow scheduling is an NP-hard problem [109, 116] which refers to the allocation of sufficient resources (human or computational) to workflow activities. The schedule impacts the workflow makespan (execution time) and cost as well as the computational resources utilisation.

In this chapter, we focus on the *Scheduler* component from the SDaaS architecture (see Figure 4.1). The scheduler is part of the enactment service of the SDaaS architecture. It is responsible for allocating activities to suitable workflow engines (which satisfy the activities' requirements) for execution. To reduce the software process execution cost in the cloud, we define the software development workflow scheduling problem and analyse the cost factors associated with cloud-based execution of such workflows. Then, we adapt three algorithms for software workflows scheduling and propose a fourth one. We evaluate these algorithms through simulation and we benchmark their performance in terms of execution cost and time. The simulation results show that our proposed algorithm saves between 19.74% and 45.78% of the execution cost and provides the best resource (VM) utilisation compared to the other presented algorithms while providing the second best makespan.

4.2 Background

Workflow scheduling in the cloud and grids has been investigated (e.g., [20, 101, 108]) where several algorithms have been proposed with different objectives (cost reduction, meeting deadlines, makespan optimisation, etc.). These algorithms have mostly focused on scientific or business process workflows and none have addressed

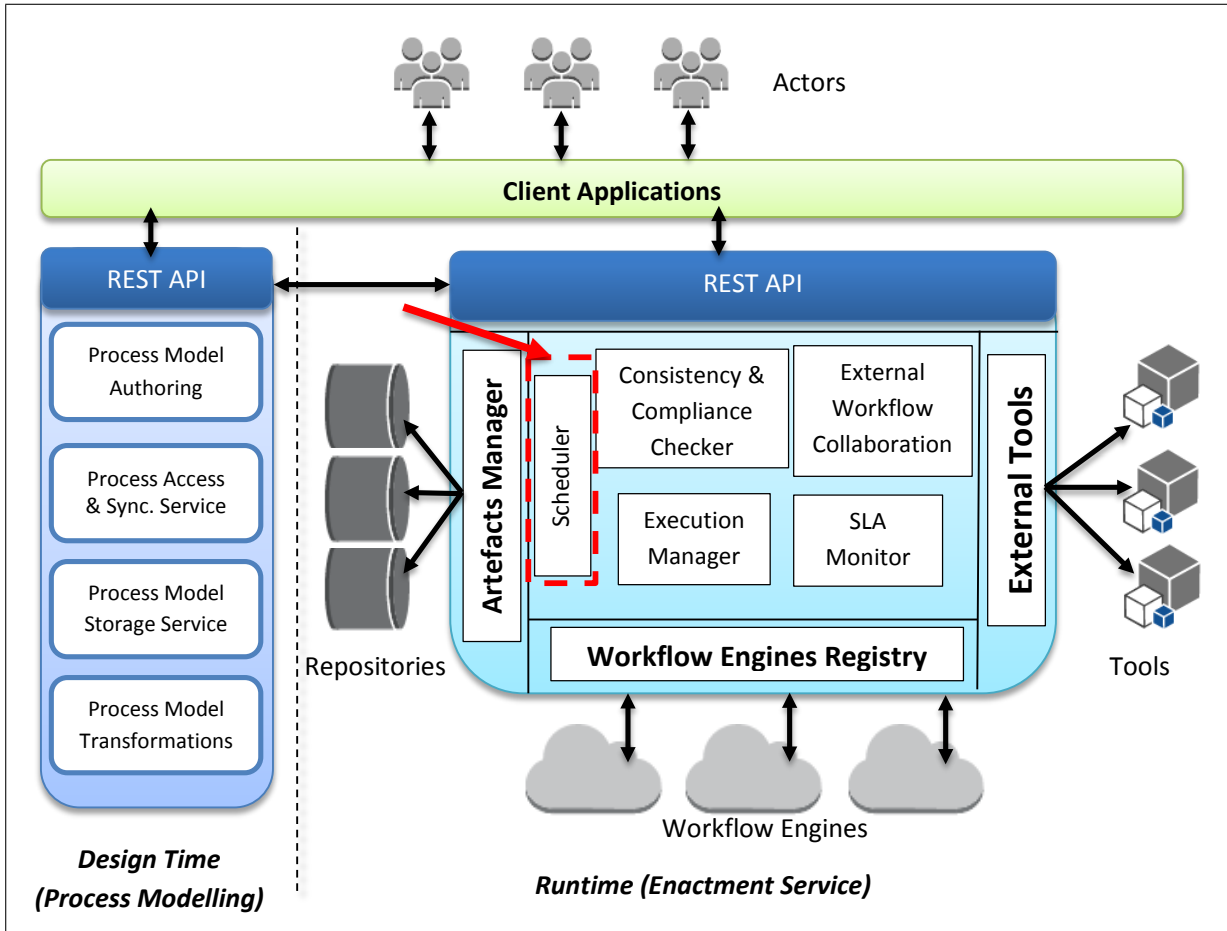


Figure 4.1: Highlighting the scheduler in the SDaaS reference architecture

software process workflows. As we have discussed in Chapter 1, software processes are a special type of business processes. They are characterised by their dynamicity, long life, interactions and control flow. Software process workflows include a diverse set of activities with different computational requirements. These activities can be interactive, control flow points or intensive computational tasks. This chapter investigates how to allocate the appropriate resources for each activity in a cost-efficient way. But first, in this section we provide some background on workflow scheduling and we review a sample of the existing scheduling algorithms.

4.2.1 Workflow scheduling

Workflow scheduling is an essential task towards the execution of workflows. Scheduling is the process of mapping sufficient resources to workflow tasks to meet some performance/QoS constraints and optimise resource utilisation. In addition, scheduling is done based on the sequence of the workflow tasks and their data and control de-

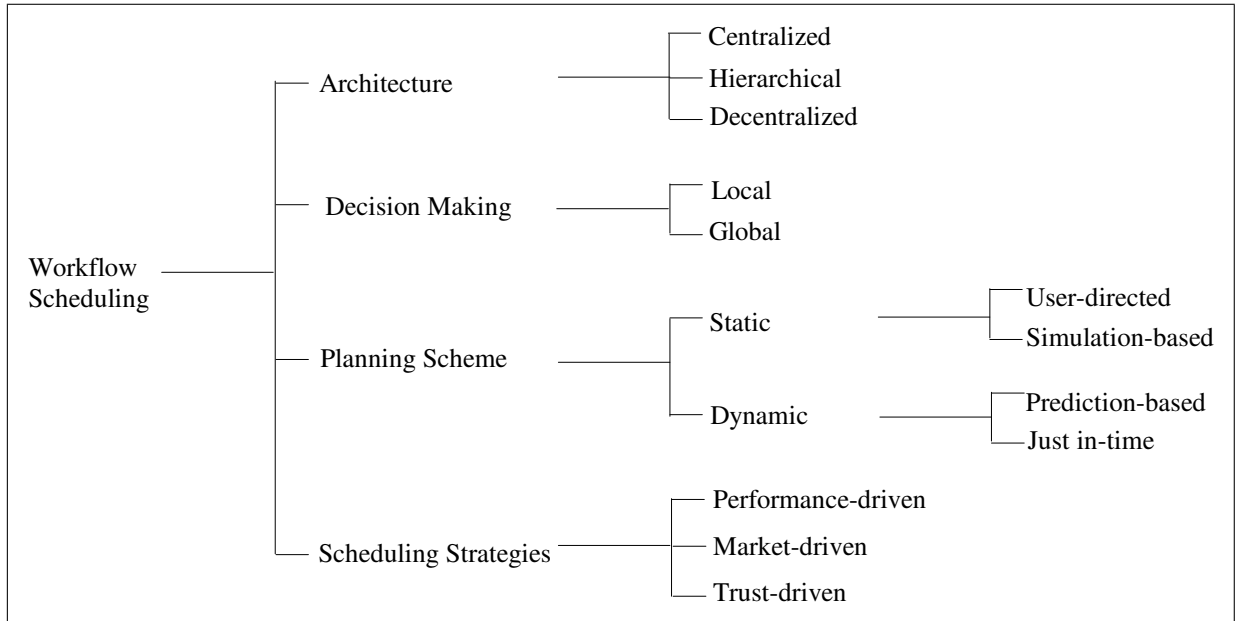


Figure 4.2: The workflow scheduling categories [116]

dependencies. A good scheduling algorithm would meet user requirements (in the form of performance/QoS constraints) and improve the resource utilisation which leads to overall WfMS performance enhancement. The performance/QoS constraints could be: cost, workflow makespan (execution time), meeting deadlines, trust and security, etc.

Workflow scheduling approaches can be categorised based on different criteria. Yu et al [116] categorised workflow scheduling based on four criteria as described in Figure 4.2. The categories are:

Architecture

Workflow scheduling is a key module in any WfMS. Therefore, the design and architecture of the workflow scheduling impacts the overall WfMS performance and evolution. As Figure 4.2 shows, there are three architectural categories for workflow scheduling:

- *Centralised*

In centralised scheduling, a single (central) scheduler is used to schedule all workflow tasks. The benefit is that the central scheduler will have full information about the workflow tasks and the available resources. This allows the scheduler to make efficient schedules. On the other hand, a central scheduler does not scale well and becomes a single point of failure,

- *Hierarchical*

Hierarchical scheduling is a multi-level scheduling where a central scheduler delegates sub-workflows to be scheduled by different lower-level schedulers. This architecture allows the central scheduler to have a different scheduling policy compared to the lower-level schedulers. In addition, this architecture scales better than the centralised one. However, the main downside is that the central scheduler still acts as a single point of failure,

- *Decentralised*

In contrast to centralised scheduling, decentralised scheduling relies on multiple independent schedulers which can communicate with each other to distribute the scheduling load between themselves. While the lack of centralised scheduler avoids having a single point of failure, none of the schedulers have all the information about the entire workflow and its tasks. The lack of the global view means that the produced schedules are unlikely to be optimal.

Decision making

Another categorisation of workflow scheduling approaches is based on the information used to make the scheduling decision. If the decision is made based on the information of the single workflow task at hand (without considering the rest of the workflow), it is called a *local decision*. Local decisions are cheap to produce but they are not usually optimal due to ignoring the entire workflow. In contrast, a *global decision* is based on the information of the entire workflow which produces more optimal schedules comparing to the local decision approach. However, this optimality comes with the cost of expensive computation and longer scheduling times.

Planning scheme

To execute a workflow, the workflow abstract model needs to be translated into a concrete model (workflow instance). The scheme of this translation is another criteria to categorise workflow scheduling approaches. The scheme can be either *static* or *dynamic*. In the static scheme, the workflow instance should be fully created before execution. The generated schedule is rigid and does not consider any real-time changes. The scheduling can be based on user's knowledge (user-directed) or based on simulating the workflow execution on a set of resources and selecting the best schedule.

On the other hand, the dynamic scheme schedules the execution in real-time allowing for considering real-time changes. Dynamic scheduling can be based on prediction results along with dynamic data (prediction-based) or can happen at the time of execution (just-in-time).

Scheduling strategies

The last criteria is based on the type of QoS requirements (strategies) constraining the execution scheduling. These strategies can be: Performance-driven (aiming to achieve optimal performance metric such as workflow makespan), Market-driven (aiming to acquire the most cost-effective services and resources to execute the workflow), and Trust-driven (aiming to choose trusted resources to execute the workflow based on their security policies for example).

4.2.2 Workflow scheduling algorithms

In this subsection, we review six state-of-the-art scheduling algorithms and their suitability for scheduling software processes in the cloud. While several authors have surveyed workflow scheduling algorithms (e.g., [20, 101, 108]), here, we review a sample of algorithms which target workflow execution cost and/or time in the cloud or grids.

A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on a Cloud Computing Platform

Liu et al. [82] proposed an algorithm for scheduling workflows with large number of instances (instance-intensive) and cost constraints on the cloud. It aims to minimise cost under user designated deadlines or minimising execution time under user designated budget. The algorithm dynamically calculates the relation between cost and execution time and visualises it to the user so that he/she can make a choice to compromise time or cost. The algorithm is compared against the Deadline-MDP algorithm [115] in terms of cost and makespan and shows that it reduces execution cost by over 15% whilst meeting the user-designated deadline and reduces the mean execution time by over 20% within the user-designated execution cost.

However, it is worth noting that the cost calculation in this algorithm does not consider the execution time taken by a task and instead uses a hard-coded table for execution

prices based on the provided processing speed. In addition, this algorithm does not support tasks to have special resource requirements such as private resources or specific computational power which is needed in software processes as we explained in Chapter 3. Additionally, the idea of applying deadlines to software processes workflows is not practical due to the fact that it is hard to predict/control the execution time of many activities in such processes. Especially, the ones which rely on human intervention.

Auto-Scaling to Minimise Cost and Meet Application Deadlines in Cloud Workflows

Mao et al. [85] proposed an algorithm for scheduling workflow tasks within a given deadline and at the minimal cost by dynamically allocating/deallocating VMs. The schedule is dynamically calculated to auto-scale VMs to handle dynamic loads from multiple workflows.

While this approach would fit for data-intensive or business process workflows, as mentioned earlier, allocating deadlines for software processes is not practical. Software processes have a mixture of human-performed and tool-supported tasks. The human-performed tasks are often unpredictable and can be long-running, therefore, it would be challenging to allocate sub-deadlines for this type of tasks.

Scaling and Scheduling to Maximise Application Performance within Budget Constraints in Cloud Workflows

In another study [84], Mao et al. proposed two algorithms to maximise performance (makespan) while meeting budget constraints. The first algorithm is: *Schedule-first* which splits the total budget onto individual jobs and finds the fastest schedule before acquiring the resources. The second algorithm is: *Scale-first* which determines the required amount and type of cloud resources and then allocate jobs to the acquired resources. Their experiments show that the *Scale-first* algorithm performs better in low budgets while the *Schedule-first* performs better in higher budgets.

While this study considers the variety of cloud resources requirements, it does not handle multiple concurrent workflow instances which are possible to happen in a software project which involves multiple stakeholders participating in multiple processes. Additionally, this study provides static schedules which does not consider the dynamicity of both cloud and software processes.

Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT)

Instead of having different scheduling objectives aggregated in one function, Durillo

et al. [44] proposed a Pareto-based approach which produces a set of (nearly) optimal trade-off solutions for users to choose from. The aim of the algorithm is to minimise the execution cost and the makespan. They compare their approach with the SPEA2* approach and show that it produces higher quality solutions.

Similar to the previous study, this approach, does not handle the dynamic load of multiple software workflows executing concurrently.

A Market-oriented Hierarchical Scheduling Strategy in Cloud Workflow Systems

In [113], Wu et al. proposed a hierarchical two-step scheduling approach to meet QoS constraints for workflow instances while minimising the execution cost. The first step is the static *Service-level Scheduling* which maps workflow instance tasks to global cloud providers based on the QoS requirements. The second step is the dynamic *Task-level Scheduling* where tasks are mapped to VMs in the local data centre of the selected cloud provider. They adapt three different algorithms (Genetic Algorithm, Ant Colony Optimisation and Particle Swarm Optimisation) to perform the second step and compare their performance. The experimental results show that the *Ant Colony Optimisation* gives the best results compared to the other two in terms of CPU time, makespan and cost.

Being a market-oriented approach, this approach does not target optimising the workflows makespan. As we explained earlier, market-oriented scheduling aims to choose the most cost-efficient resources. However, the workflow makespan is an important aspect of a software process. Especially, when a process is prioritised for execution as other processes are dependent on it.

Adaptive workflow scheduling for dynamic grid and cloud computing environment

Rahman et al. [94] proposed the dynamic critical path scheduling approach for grids (DCP-G). This proposal aims to improve the scheduling performance in the dynamic resource environment in grids. It does that by dynamically mapping workflow tasks to grid resources based on calculating the critical path in the workflow task graph. In addition, the authors propose an adaptive workflow management approach for data analysis workflows in hybrid clouds.

This approach aims to meet users QoS constraints such as execution time and budget. However, it does not consider software process workflows which are more complex and include different types of tasks. Furthermore, although this approach aims to meet budget constraints but it does not consider reducing the execution cost of multiple concurrent workflow instances execution.

Adaptive Task Schedule

In [109], Wang et al. proposed a dynamic adaptive task schedule algorithm which dynamically sets a maximum number of VMs that can be acquired at any given time. This limit is calculated based on two variables: either historical (backward) or future (forward) number of tasks and an arbitrary threshold. They compare this algorithm with three other algorithms (one static and two dynamic) and their results show that the adaptive task schedule algorithm based on future number of tasks gives the best performance.

This algorithm, however, does not handle specific requirements of each workflow task (which is needed for software processes as we discussed earlier) and relies on an arbitrary value which does not have any rules to calculate. In Section 4.3.7.3, we will adapt this algorithm to the SDaaS architecture needs and we show that our proposed algorithm outperforms this one.

Summary. The existing scheduling approaches have been focusing on scientific (data analysis) workflows which differ from software processes as we have seen in Chapter 1. Some approaches use static scheduling mechanism that does not handle the dynamicity and heterogeneity of cloud resources. Other approaches adopt dynamic scheduling mechanisms and target to meet one or more optimisation criteria (makespan, cost, budget, deadline). In addition, few approaches considered the diverse requirements that different workflows tasks may require in terms of cloud resource types. To the

best of our knowledge, no existing research has addressed scheduling software process workflows in the cloud and the catering for the special needs of such workflows. The next section highlights those needs and introduces the proposed scheduling algorithms.

4.3 Scheduling SDaaS Software Workflows in the Cloud

Workflow scheduling is an NP-hard problem [109]. The SDaaS architecture needs to schedule software workflows execution in the cloud. In this section, we set the assumptions and objectives which motivate the scheduling process before we propose a scheduling algorithm.

4.3.1 Assumptions

The following assumptions describe the scenarios in which the SDaaS architecture will be used:

- The SDaaS architecture will be used by an organisation which have multiple (geographically-distributed) teams which collaborate on several projects concurrently.
- Software processes contain a set of activities with different requirements for execution privacy and computational resources.
- Some activities may be required to be performed quickly while others may not. In a delayed project, certain activities will be required to be performed quickly to avoid further delays. In addition, critical activities that precede the execution of many other activities are naturally expected to be performed faster so that they do not block other activities longer. These activities are referred to as *priority activities*.
- Interactive activities are not executed on the cloud since these activities may involve stakeholders performing certain tasks offline. Likewise, scheduling the human activities (the ones preformed solely by humans without any tool support) is not considered since it does not have an impact on the cost of using the cloud.

- An activity becomes ready for execution once all of its input artefacts become available.
- At any given time, there might be several ready-to-execute activities from different processes.
- Activities execution times are presumed to be known. Execution time estimation techniques are available (e.g., [71, 103]) but are out of the scope of this chapter.
- The cost of executing an activity is dependent on the time it takes to finish and the cost of data transfer outside the cloud provider boundary. For simplicity, both the public and the private cloud resources are assumed to be located within two data centres (one public and one private) and data transfer between them is negligible. Therefore, data transfer costs are assumed to be negligible.

4.3.2 Objectives

The objectives of the scheduling software workflows in the SDaaS architecture are:

1. To allocate activities to a workflow engines pool containing engines which match the required resources by the activity.
2. To reduce the overall workflows cloud-based execution cost by switching workflow engines on/off when needed/unneeded.
3. Reducing the cost conflicts with the workflow makespan (execution time). The scheduling should minimise the impact of reducing the cost on the workflow makespan.
4. To utilise the available workflow engines as best as possible.

4.3.3 Motivation

Given the assumptions and objectives of the scheduling process, we notice that software workflows in the SDaaS architecture have specific needs that are not addressed by the existing scheduling algorithms (see Section 4.2).

Unlike scientific workflows, software processes are control-flow workflows which involve more human interactions. Furthermore, different types of software processes tasks can require different types of resources in terms of computational power and/or deployment choices (public vs private clouds). These requirements include the choice of public or private cloud, cloud provider (in case of using public clouds), the virtual machine image, machine type (specifying the amount of memory, CPU power and network bandwidth) and number of machines (in case of a distributed activity).

The SDaaS architecture presented in Chapter 2 allows modelling and executing software process workflows on a number of distributed workflow engines. The process models describe the resources requirements for each activity in the process. As we mentioned in Section 4.1, the feeling of having unlimited pool of resources in the cloud is illusional. Therefore, unwise use of cloud can result in huge costs. In this section, we propose an algorithm to schedule software process workflow execution in the cloud with the aim of cost reduction without increasing the workflow makespan (execution time).

Before we dive into the proposed scheduling algorithms, in the next subsection, we will define the scheduling problem and set some assumptions. Then, based on these definitions and assumptions, we will elicit a set of requirements that the scheduling algorithm should satisfy.

4.3.4 Problem definition & assumptions

As any other scheduling problem, the target is to map workflow activities to the right resources in order to achieve some improvement (e.g., on performance or cost). Here, we formally define the elements related to the scheduling problem. These elements fit into two main categories: (a) the cloud resources model and (b) the workflow model.

The cloud resources model

Cloud resources include compute, storage and networking solutions. These solutions are used to power workflow engines which execute workflow activities.

Definition (1) Workflow Engine (WE): is a software service deployed on a cloud VM and it hosts the execution of workflow activities.

$$WE = (MachineType, HostType, State) \quad (4.1)$$

where *MachineType* is the set of available VM machine types and *HostType* is defined as:

$$HostType = \{public, private\} \quad (4.2)$$

The workflow *State* is the set of operational states for the engine and is defined as:

$$State = \{active, inactive\} \quad (4.3)$$

Each workflow engine can execute a single workflow activity at any given time.

Definition (2) Workflow Engines Pool (WEP): is the set of workflow engines that have the same *MachineType* and *HostType*.

$$WEP_i = (WES_i, MachineType, HostType, R_i) \quad (4.4)$$

where R_i is the pool size limit (maximum number of active workflow engines in the pool) and WES_i is the set of workflow engines in the pool. WES_i is defined as:

$$WES_i = \bigcup_{j=1}^n \{WE_j \mid WE_j.MachineType = WEP_i.MachineType \wedge WE_j.HostType = WEP_i.HostType\} \quad (4.5)$$

The workflow model

Software process workflows belong to software development projects.

Definition (3) A software development project (P) contains multiple workflows and is defined as:

$$P = \bigcup_{i=1}^n \{W_i\} \quad (4.6)$$

where W_i is workflow number i . Multiple teams can be involved in a single project and might execute multiple workflow instances concurrently.

Definition (4) An activity is the smallest unit of execution in the workflow and is defined as:

$$A_i = (T, Req, priority, ET) \quad (4.7)$$

where T is the executable task, Req is the resources requirements (machine and host types) and since some activities may be required to be performed quickly while others may not, $priority$ denotes whether the activity is urgent or not (an urgent activity is referred to as *priority activity*). ET is the execution time for the activity. This is assumed to be known beforehand. Although execution time estimation techniques are available (e.g., [71, 103]), they are out of the scope of this chapter.

Definition (5) A workflow is a set of activities (A) and their control and data dependencies. It is defined as:

$$W_i = (A, D) \quad (4.8)$$

where A is the set of activities and is defined as:

$$A = \bigcup_{i=1}^n \{A_i\} \quad (4.9)$$

and D is the set of dependencies which is defined as:

$$D = \{(A_i, A_j) \mid (A_i, A_j \in A \times A)\} \quad (4.10)$$

Activities in the workflow can be executed only when their predecessors if any have finished executing. Predecessors $Pred(A_i)$ are defined as:

$$Pred(A_i) = \{A_k \mid (A_k, A_i) \in D\} \quad (4.11)$$

The set of successors $Succ(A_i)$ is defined as:

$$Succ(A_i) = \{A_k \mid (A_i, A_k) \in D\} \quad (4.12)$$

The workflow execution can follow one or more of the available successors.

4.3.5 Scheduling requirements

Based on the definitions and assumptions discussed above, the scheduling must meet the following requirements:

- Since activities in the process have varying resource requirements (as discussed in Section 4.3), the scheduler should allocate activities to workflow engines which satisfy these requirements.
- Since multiple processes can be executing at the same time and multiple activities can be ready to execute at a given time, the scheduling should be **dynamic** see Section 4.2.
- The generated schedule should allocate activities to workflow engines for execution in a cost-efficient way. This means reducing the cost of using cloud resources by making the best possible use of each running workflow engine before switching it off and by having a policy for scaling the number of workflow engines up and down based on the expected load. Producing such schedules requires global knowledge of workflows being scheduled in the system. Therefore, the scheduling should be **centralised** [116]. The down side is that the scheduler scalability will be limited. Additionally, the decision making should be **Global** see Section 4.2.
- The generated schedule should minimise the overall execution time of a process while reducing the execution cost by switching off unneeded workflow engines

and optimising the use of the available ones. For those activities requiring speedy execution, an exception should be made to allow faster execution even though that might increase the execution cost. Therefore, the scheduler should balance between a **performance-driven** and a **market-driven** strategy.

4.3.6 *Cost factors*

The cost for cloud-based software workflows execution is mainly the cost of using cloud resources. The main cloud resource that will be used for executing software processes is virtual machines (VMs) which host workflow engines. Most cloud providers (e.g., Amazon ¹) charges per partial hour use of VMs.

In order to reduce the software workflows execution cost in the cloud, we need to understand the factors that have an impact on it and which of them we can control. These factors are listed below:

1. The variety and types of the required VMs since different machine types have different prices.
2. The number of priority activities. Priority activities bypass any limiting restrictions on creating new VMs thus potentially raising the execution cost.
3. The size and complexity of the workflow. The larger the workflow the more it will cost to execute it. In addition, the complexity of the workflow structure (in terms of forks, parallel activities and dependencies) impacts the execution cost and makespan.
4. The complexity of individual activities in workflows. This can be expressed by the execution time for the activity.
5. The concurrency and frequency of incoming workflow execution requests. This affects the load that the scheduler has to handle and can potentially create more demand on certain type of resources which will impact both the cost and the waiting times for limited resources to become available.

¹www.aws.amazon.com

6. Resources acquisition constraints. If the used scheduling algorithm acquires resources without any restrictions, it will potentially cost more than an algorithm which limits the resources that can be acquired during a given period. In such case, the choice of the limit will impact the execution cost and makespan.

Since incoming workflow execution requests cannot be controlled, most of these factors are uncontrollable. The only controllable factor is the resources acquisition constraints. The scheduling algorithms could apply some constraints which limit the amount and optimise the use of the acquired cloud resources. This would potentially reduce the execution cost. The next subsection describes four different scheduling algorithms for scheduling software workflows execution in the cloud.

4.3.7 *Scheduling algorithms*

The scheduling needed for software workflows is a multi-criteria scheduling which aims to meet the execution requirements of each activity and reduce the overall execution cost (of all workflows) while not significantly increasing the execution time (of individual workflows). Here, we define the terms related to the scheduling algorithms:

- Workflow engines pool: is a pool of workflow engines deployed on similar virtual machines (in terms of computational power and deployment model).
- Workflow makespan (execution time): is the difference between the execution start time of the first activity in the workflow and the execution end time of the last activity in the workflow.
- Workflow engine operational hours: are the hourly units of time starting from the time a workflow engine starts.
- Workflow engines pool size (R): is the maximum number of active workflow engines a pool can have at any given operational hour.
- Execution cost: is the cost of executing all the desired workflows in the SDaaS architecture. This can be calculated by aggregating the cost of running each workflow engine instance as follows:

$$Cost = \sum_{i=1}^n VM_n * t_n \quad (4.13)$$

Where VM_n is the price per partial hour for running the virtual machine hosting the workflow engine and t_n is the number of partial hours that workflow engine has been running.

Workflow engines are deployed on virtual machines (VMs) in the cloud. Most cloud providers charge per partial hour usage of VMs. This means that the usage time is rounded to the ceiling number of hours. For example, a one hour and ten minutes usage is charged as two hours. Therefore, to achieve cost reduction, the workflow engines pool size R for a given pool should be limited and the workflow engines in the pool should be utilised as best as possible before they are shut down. For example, if a workflow engine becomes idle after executing a 10 minutes activity, it can be kept on standby for the next 50 minutes (to accommodate any upcoming activities) without incurring any extra cost. To illustrate the effect of activities allocation to workflow engines on the execution cost, let us have a look at Figure 4.3. The figure shows three activities [A1, A2, A3] and their execution times [20, 40, 20] minutes respectively. Each activity is allocated to a workflow engine resulting in the cost of three partial hours and underutilised workflow engines (the grey areas representing 100 minutes of idle time). While in Figure 4.4, the three activities are allocated on the workflow engine resulting in two partial hours cost and better utilisation of the workflow engine (40 minutes of idle time). The latter scenario would be ideal if the three activities were sequential. However, if they were concurrent, some of the activities will wait for others to finish executing. Therefore, there would be a trade-off between the workflow execution cost and makespan.

Since the pricing for VMs is per partial hour, then starting and shutting down VMs should happen at the beginning of each operational hour. The decision to allocate an activity to a workflow engine should be made only when the activity becomes ready to execute, i.e., it is an event-driven decision. In this chapter, we try four different scheduling algorithms and benchmark their performance from both execution cost and makespan perspectives.

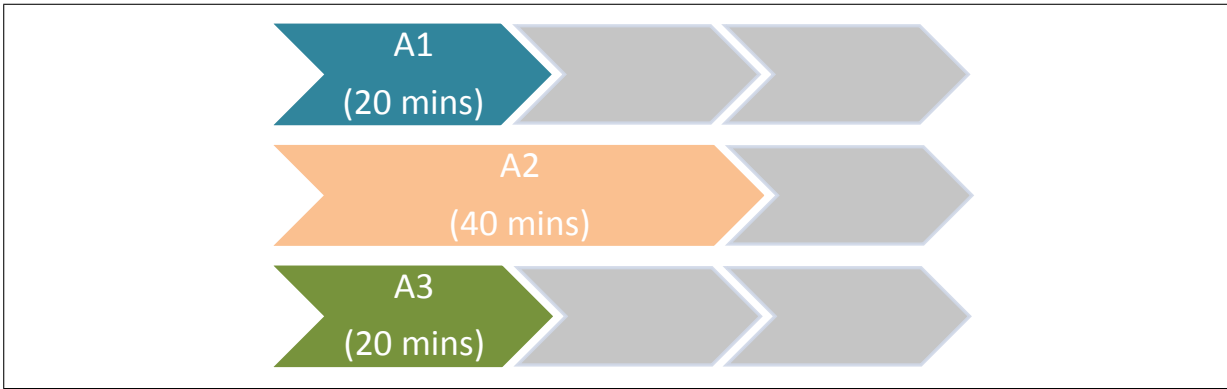


Figure 4.3: Allocating activities to workflow engines (a)

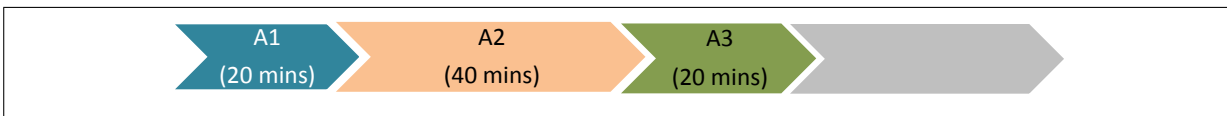


Figure 4.4: Allocating activities to workflow engines (b)

Below we explain the four algorithms. These algorithms are all dynamic and centralised.

4.3.7.1 Unlimited First Come First Serve (UFCFS)

This is the simplest and most basic scheduling approach where the pool size R is always set to infinity. Once an activity is ready-to-execute, it is allocated to an available workflow engine in the relevant workflow engines pool (if exists), otherwise a new pool and/or workflow engine are created. Figure 4.5 shows the *UFCFS* algorithm.

4.3.7.2 Limited First Come First Serve (LFCFS)

This is a similar approach to the UFCFS except that there is a universal limit on the number of active workflow engines in any workflow engines pool at any time. Figure 4.6 shows the LFCFS algorithm. The workflow engines pool size limit (R) is an arbitrary value which aims to restrict the execution cost. If all workflow engines in a pool are busy and their number has reached R and a new activity is ready to be executed in this pool, the scheduler will allocate this activity to the workflow engine with the earliest finishing time. This means that the activity will be delayed until a suitable workflow engine becomes available again.


```

1 Activity A;
2 List<WorkflowEnginePool> pools;
3
4 start
5 find a pool in pools which match the computational resources and privacy
   requirements of A.
6 if(pool is found)
7 {
8     find an available workflow engine
9     if(workflow engine is found)
10        add A to the jobs queue of the engine;
11    else
12    {
13        create and start a new workflow engine and add A to its jobs queue;
14    }
15 }
16 else
17 {
18     create a pool;
19     create and start a new workflow engine in the new pool and add A to its
   jobs queue;
20 }
21 end

```

Figure 4.5: Unlimited First Come First Serve algorithm

4.3.7.3 Pool-based Adaptive Task Schedule

This algorithm is adapted from the Adaptive Task Schedule algorithm [109] described in Section 4.2. Here, we define a workflow engines pool size limit R dynamically for each pool at the beginning of each operational hour, hence the name *Pool-based*. the algorithm consists of two main steps:

1. Matching each ready-to-execute activity with a suitable workflow engines pool (a pool which contains workflow engines matching the required resources for the activity).
2. For each workflow engines pool i , the pool size limit R_i is dynamically calculated using the following formula:

$$R_i = T * E_i \quad (4.14)$$

Where T is a universal arbitrary real value between 0 to 1 which indicates the proportion between the activities to be executed and the workflow engines. For example, when T

```

1 Activity A;
2 List<WorkflowEnginePool> pools;
3 int R; // the max number of workflow engines in each pool
4
5 start:
6 find a pool in pools which match the computational resources and privacy
   requirements of A.
7 if(pool is found)
8 {
9     find an available workflow engine;
10    if(workflow engine is found)
11        add A to the jobs queue of the engine;
12    else
13    {
14        if(number of workflow engines in pool  $i < R$ )
15            create and start a new workflow engine and add A to its jobs
16            queue;
17        else
18            allocate A to the first available engine;
19    }
20 else
21 {
22     create a pool;
23     create and start a new workflow engine in the new pool and add A to its
24     jobs queue;
25 }
end

```

Figure 4.6: Limited First Come First Serve algorithm

is 0.5, it means that there should be a workflow engine for each two activities. E_i is the number of activities which match pool i and are expected to start in the next hour.

Unlike the original algorithm which has two versions (one looking forward and one backward), here we only look at the expected activities in the next hour (forward). Since the activities arrive in a non-deterministic way, the history alone does not necessarily give an accurate prediction for the predicted load in the next hour.

Figure 4.7 shows this algorithm. As we can see, the algorithm is very similar to the LFCFS algorithm except that each pool has its own R .

4.3.7.4 Proportional Adaptive Task Schedule

Similar to the previous two algorithms, this algorithm sets a limit for the workflow engines pool size R . The difference is that R is now calculated based on the proportion between the execution time of the activities that are predicted to start in the next hour and those which have started execution in the past hour. The following formula is

```

1 Activity A;
2 List<WorkflowEnginePool> pools;
3 List<int> R; // the max number of workflow engines for each pool
4
5 start:
6 find a pool in pools which match the computational resources and privacy
   requirements of A.
7 if(pool is found)
8 {
9     find an available workflow engine;
10    if(workflow engine is found)
11        add A to the jobs queue of the engine;
12    else
13    {
14        if(number of workflow engines in pool  $i < R_i$ )
15            create and start a new workflow engine and add A to its jobs
   queue;
16    else
17        allocate A to the first available engine;
18    }
19 }
20 else
21 {
22     create a pool;
23     create and start a new workflow engine in the new pool and add A to its
   jobs queue;
24 }
25 end

```

Figure 4.7: Pool-based Adaptive task scheduling algorithm adapted from [109]

applied when R_i for a given pool i is calculated for the first time:

$$R_i = \left\lfloor \frac{T_{next}}{60} \right\rfloor \quad (4.15)$$

Where T_{next} is the total execution time of the activities that will start in the next hour (in minutes). Therefore, R_i is the floor of the expected execution hours needed to execute the activities that would start in the next hour. When R_i has been set before, the following formula is applied to calculate R_i on every operational hour:

$$R_i = \left\lceil \frac{T_{next}}{T_{past}} * R_i' \right\rceil \quad (4.16)$$

Where T_{past} is the total execution time (in minutes) of the activities that have started in the past hour and R_i' is the last value of R_i . The proportional adaptive task schedule

algorithm itself is the same as the pool-based adaptive task schedule algorithm in Figure 4.7.

4.4 Evaluation

To analyse the performance of the algorithms described in the previous section, we simulate the execution of each algorithm and measure two metrics (as defined in Section 4.3.7): (a) the makespan for each simulated workflow, and (b) the total cost of executing all workflows. The simulation is implemented in Java where the scheduler uses one of the four algorithms to schedule activities from multiple workflow instances. In this section, we describe the set up and configuration parameters for the simulation. The simulation consists of three main components: (a) the request generator, (b) the simulation scheduler, and (c) the workflow engines.

4.4.1 *The request generator*

In order to simulate a real workflow execution scenario, the request generator generates requests to execute workflow instances at random times to create non-determinism. In a real scenario, workflow instances can be requested to be executed at any time and might be executing in parallel with some other instances.

Since randomisation is used, there is a need to run the simulation several times and calculate mean values for the desired metrics. We use three input workflow models of sizes 7, 9 and 10 activities. These models have different requirements for activities (a mixture of public/private and priority/non-priority activities). The structure of these models and which activity has which requirements are irrelevant as the request generator randomly chooses a time to trigger the request for each of the three input models in each simulation iteration. This creates a non-deterministic load on different computational resources.

Although these input models are random, they reflect the possible incoming software workflows execution requests. As we explained in Section 4.3.1, in the SDaaS architecture, multiple workflow instances can be executing concurrently. The demand on different workflow engines is non-deterministic as different activities (with different requirements) from different workflows can become ready-to-execute at anytime.

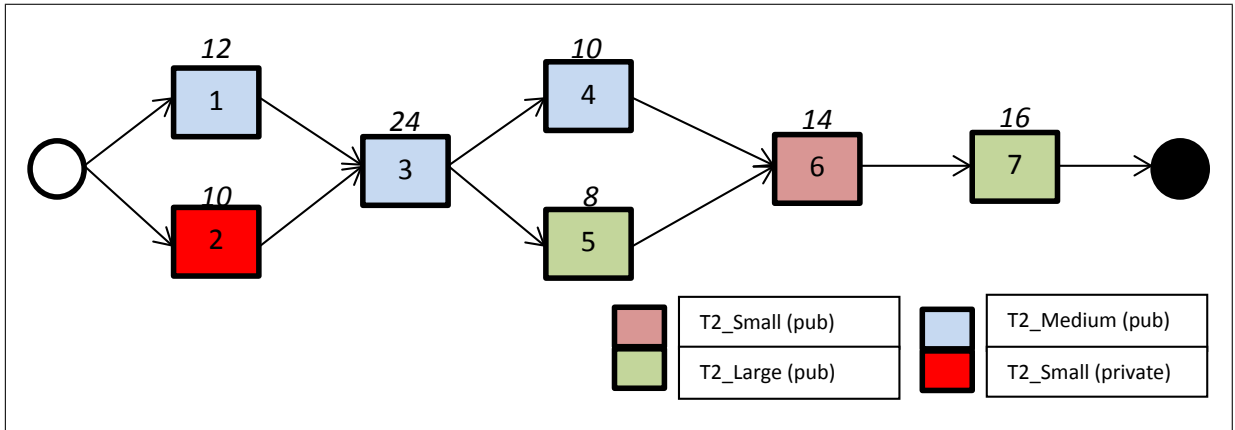


Figure 4.8: The first workflow input model

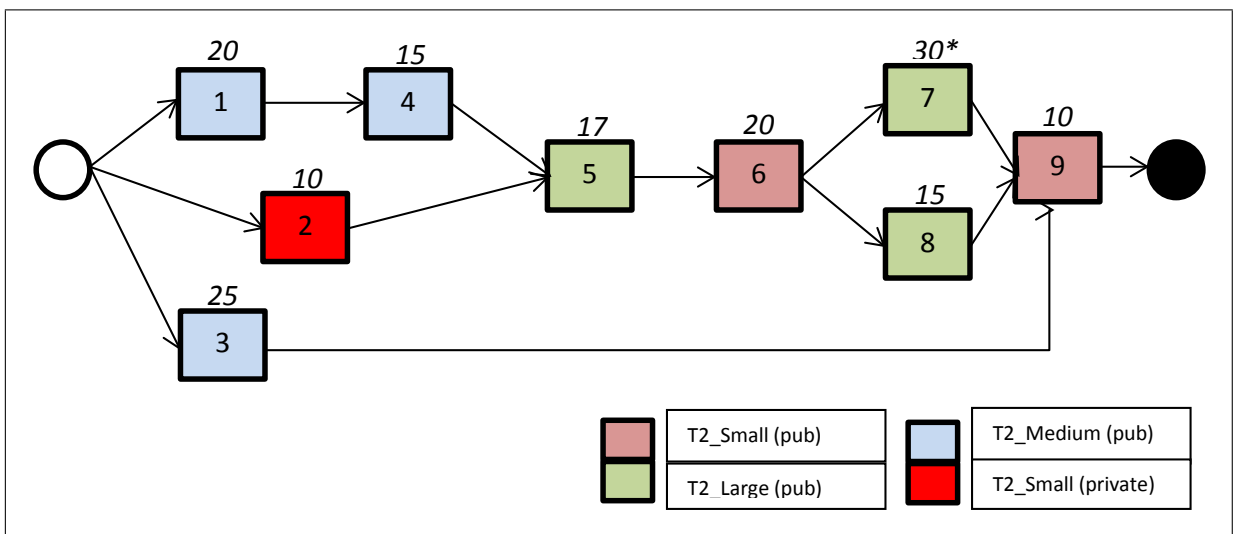


Figure 4.9: The second workflow input model

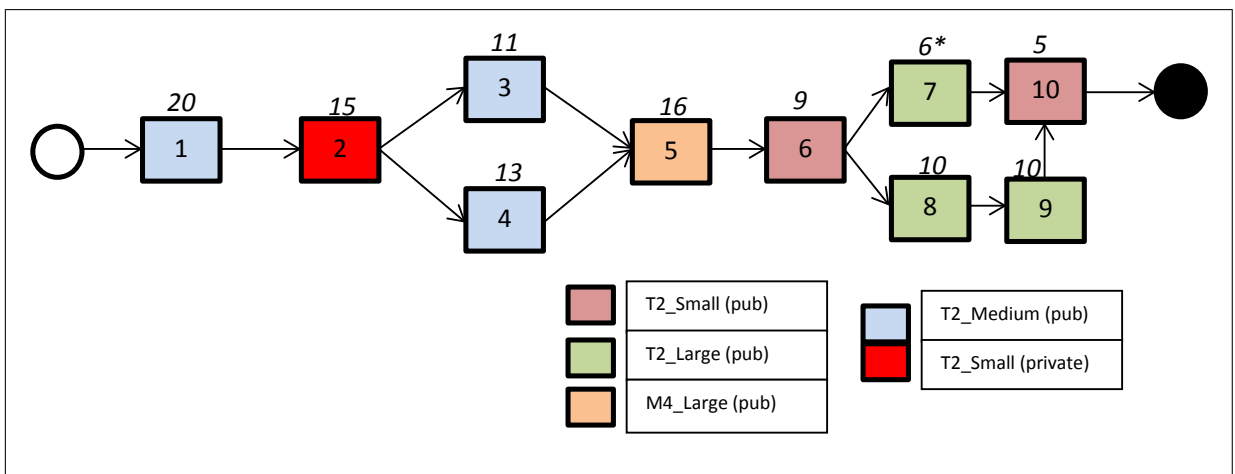


Figure 4.10: The third workflow input model

Figures 4.8, 4.9 and 4.10 illustrate the three input workflow models. Each activity is colour coded to identify the machine type it requires. The execution time is specified above each activity where a * symbol indicates that the activity is a priority activity.

Table 4.1: Workflow engine VM types and prices

EC2 Machine Type	Amazon (Public) Price (\$)	Private Price (\$)
T2_SMALL	0.026	0.0286
T2_MEDIUM	0.052	0.0572
T2_LARGE	0.104	0.1144
M4_LARGE	0.12	0.132

4.4.2 The simulation scheduler

Once the request generator triggers the execution of a workflow instance, the activities in that instance are scanned and any ready-to-execute activity is pushed into a shared jobs queue. The scheduler monitors this queue and schedules each activity to a suitable workflow engine. The scheduler takes two mandatory parameters: *the number of simulations to be run* and *the algorithm chosen for scheduling*. It also takes two optional parameters: *the limit* and *the threshold*. These two parameters are only applicable for the LFCFS and pool-based adaptive task schedule algorithms respectively.

4.4.3 Workflow engines

The workflow engines are where the execution of activities takes place. For the purpose of this evaluation, we are only concerned about how long it takes to execute an activity. Since monitoring the real-time is important here to aggregate waiting times and execution times for all activities, the workflow engine simulator simulates the execution by clock ticks. However, while activity execution times in reality can be in minutes or hours, the simulator scales the execution time down by a factor of 60. Therefore, a 60 minutes execution will be simulated as one minute. Workflow engines are hosted on VMs. Thus, the execution cost would be the product of the number of partial hours consumed and the price of the VM. In this simulation, we use a subset of Amazon EC2 VM pricing. Table 4.1 shows the list of the VM types used and their prices as offered by Amazon in the US-East region. While Amazon prices are for public cloud VMs, we assume that a private version of those VMs with the same specifications would cost 10% more than their public counterpart. This is because private cloud requires in-house hardware and software maintenance, power, cooling, etc.

4.4.4 *Performing the simulation*

As mentioned before, we need to run the simulation several times to normalise the obtained results. We run the simulation 8 different times (with different configurations) and each run consisted of 500 repetitions. The UFCFS algorithm is run once while the LFCFS is run three times (with the following values for the *limit*: 1,2 and 4). The pool-based adaptive task schedule algorithm is also run three times (with the following values for the *threshold*: 0.33, 0.5 and 0.75). And the proportional adaptive task schedule algorithm is run once. The results of the simulation are discussed in the next subsection.

4.4.5 *Simulation results*

Here, we report on the simulation results. During the simulation, the execution time of each individual workflow instance and the overall execution cost of all three instances were captured in each simulation run. Since we scaled the times down by a factor of 60 (as explained in the Section 4.4), we scale the recorded execution times up by the same factor. In addition, we need to calculate the mean value of all the 500 simulations. The simulation results are summarised in Table 4.2 where the mean execution time (in minutes) of each input workflow is presented along with the mean overall execution cost of the three workflows and the mean of the number of VMs used for execution. In addition, l and t represent limit and threshold respectively. We can notice that (expectedly) the *UFCFS* algorithm gives the fastest execution but also the most expensive one. On the other hand, the *Proportional Adaptive Task Schedule* algorithm gives the best cost efficiency (23.3% cheaper than UFCFS), the best VM utilisation and the second best overall execution time performance. Figure 4.11 shows a comparison between algorithms (and their parameter variation) in terms of execution cost.

We calculate the confidence intervals for the means of the results presented in Table 4.2 using a confidence level of 95%. The confidence levels for the cost and the number of VMs are presented in Table 4.3.

Figures 4.12, 4.13 and 4.14 show the benchmark of all algorithms (the best performing parameter in case of LFCFS and Pool-based Adaptive Task Schedule) for each input workflow model. As these charts show, for workflow models 2 and 3, the *Proportional Adaptive Task Schedule* gives the second best execution time (after the UFCFS). For

Table 4.2: Simulation results summary

Algorithm	Parameters	Execution time (W1)	Execution time (W2)	Execution time (W3)	Cost (\$)	VM No.
UFCFS	N/A	88.72	139.89	131.22	1.59	9.43
LFCFS	l = 1	200.43	253.26	208.46	1.52	5.88
LFCFS	l = 2	146.14	206.33	181.80	1.90	8.684
LFCFS	l = 4	125.58	194.65	170.89	2.25	11.00
Pool-based Adaptive	t = 0.33	193.93	254.03	208.13	1.64	5.83
Pool-based Adaptive	t = 0.5	176.09	233.25	201.59	1.76	6.56
Pool-based Adaptive	t = 0.75	165.18	217.07	193.51	1.81	7.28
Proportional Adaptive	N/A	144.06	184.15	147.19	1.22	5.81

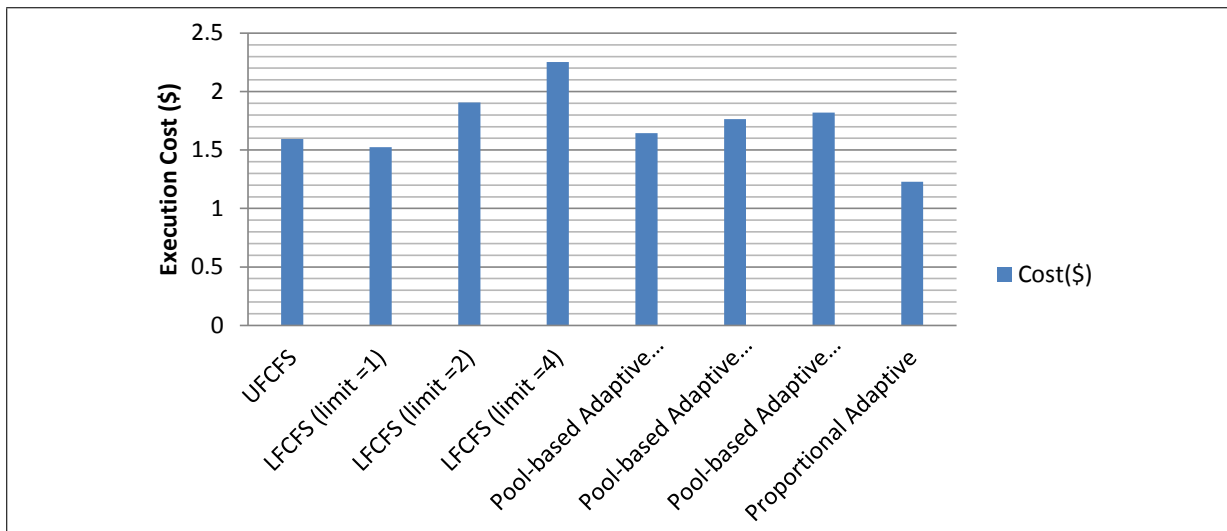


Figure 4.11: Execution cost benchmark in all algorithms

workflow model 1, this is not the case. This could be linked to the fact that workflow model 1 has relatively shorter execution time. However, more experiments are required to prove that the Proportional Adaptive Task Schedule algorithm is not the best option for short workflows. In the following subsections we detail the results further for each algorithm.

4.4.5.1 UFCFS

Figure 4.15 shows the normalised mean values for execution times of the three workflow input models. Normalisation (which is applied to most of the following charts) is achieved by dividing each value by the minimum value in its category. In this chart

Table 4.3: Simulation results with confidence intervals

Algorithm	Parameters	Cost(\$)	Confidence Interval of Cost	VM No.	Confidence Interval of VM No.
UFCFS	N/A	1.59	0.000593	9.43	0.004222
LFCFS	$l = 1$	1.52	0.00067	5.88	0.00132
LFCFS	$l = 2$	1.90	0.000859	8.684	0.002467
LFCFS	$l = 4$	2.25	0.001034	11.00	0.004358
Pool-based Adaptive	$t = 0.33$	1.64	0.000829	5.83	0.001275
Pool-based Adaptive	$t = 0.5$	1.76	0.000943	6.56	0.001948
Pool-based Adaptive	$t = 0.75$	1.81	0.0009	7.28	0.002232
Proportional Adaptive	N/A	1.22	0.000596	5.81	0.001334

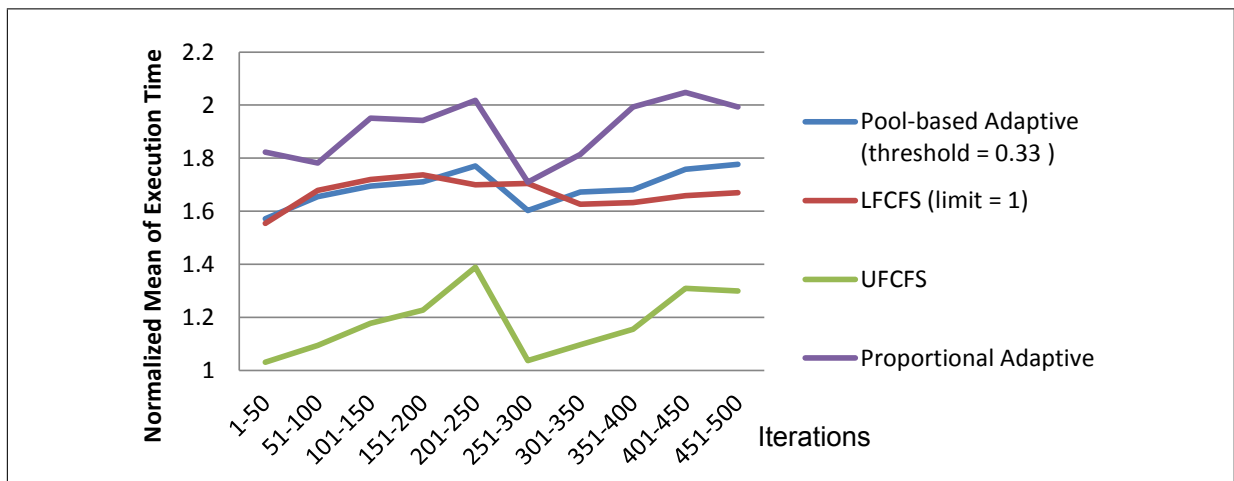


Figure 4.12: Execution time benchmark for all algorithms for workflow 1

(as well as other charts in this chapter), the simulation runs have been grouped into groups of 50 runs and their mean was calculated. As the figure shows, UFCFS provides relatively low execution time since there are no delays required. However, the execution cost and the number of virtual machines used is relatively high as shown in Table 4.2.

4.4.5.2 LFCFS

We simulated LFCFS with three different *limit* values. Figures 4.16, 4.17 and 4.18 show the execution time of the three workflows with the different *limit* values.

As we can see in these charts, arbitrarily choosing the best value for the *limit* parameter is not possible as different values perform differently. The input models and their

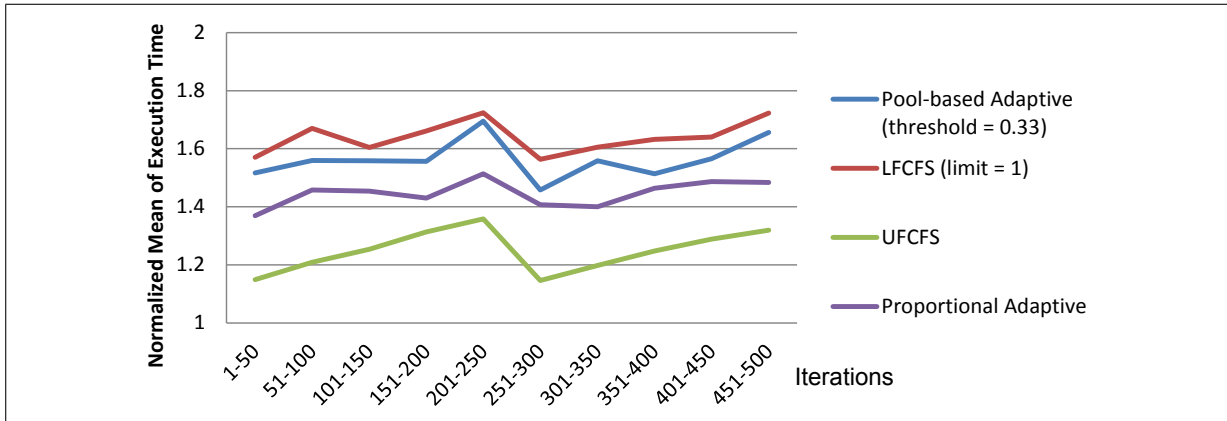


Figure 4.13: Execution time benchmark for all algorithms for workflow 2

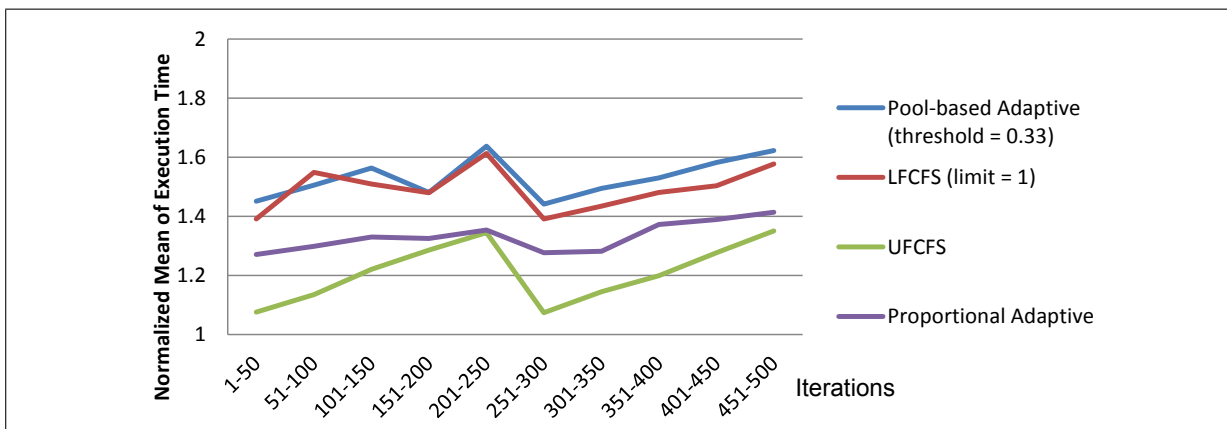


Figure 4.14: Execution time benchmark for all algorithms for workflow 3

structure and complexity are among several factors that impact the results when using a particular *limit* value. Such factors are unpredictable, therefore, there is no systematic way for deciding the best arbitrary *limit* value to use. Finally, Figure 4.19 shows the normalised mean execution time for each workflow under the three different *limit* value as well as the normalised execution cost. We can clearly see that the cost increases linearly as the *limit* increases. In contrast, the execution times are reduced when the *limit* is higher.

4.4.5.3 Pool-based Adaptive Task Scheduling

By looking at the execution cost in Table 4.2 we see that the lower the *threshold*, the lower the execution cost. In Figure 4.20, we show the mean execution time for each workflow under different *threshold* values. We also show the overall execution cost. As expected, the higher the *threshold*, the faster and more expensive the execution. But again, there is no precise mechanism for finding the right trade-off point which also depends (in real situation) on unpredictable input workflow models.

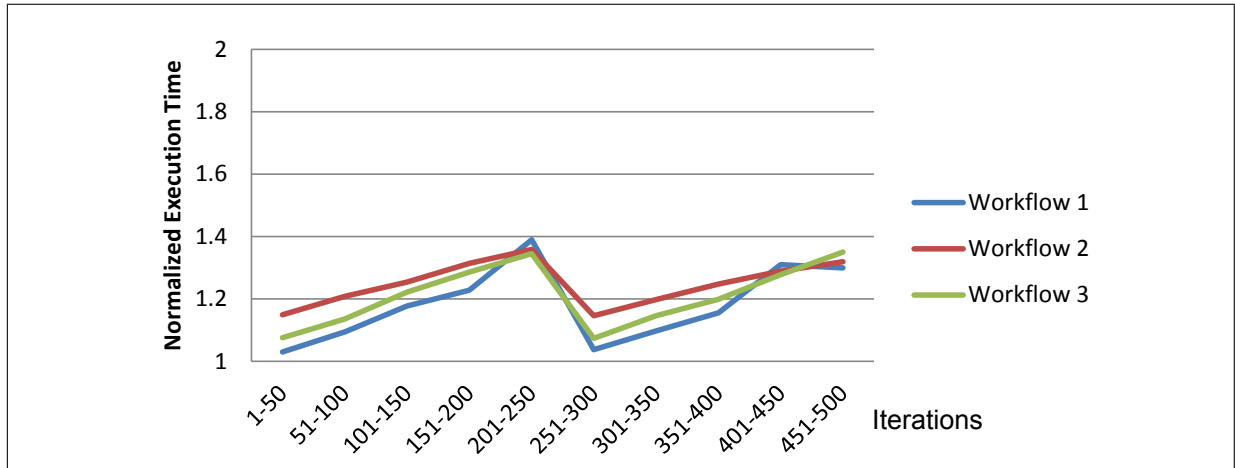


Figure 4.15: Execution times in UFCFS

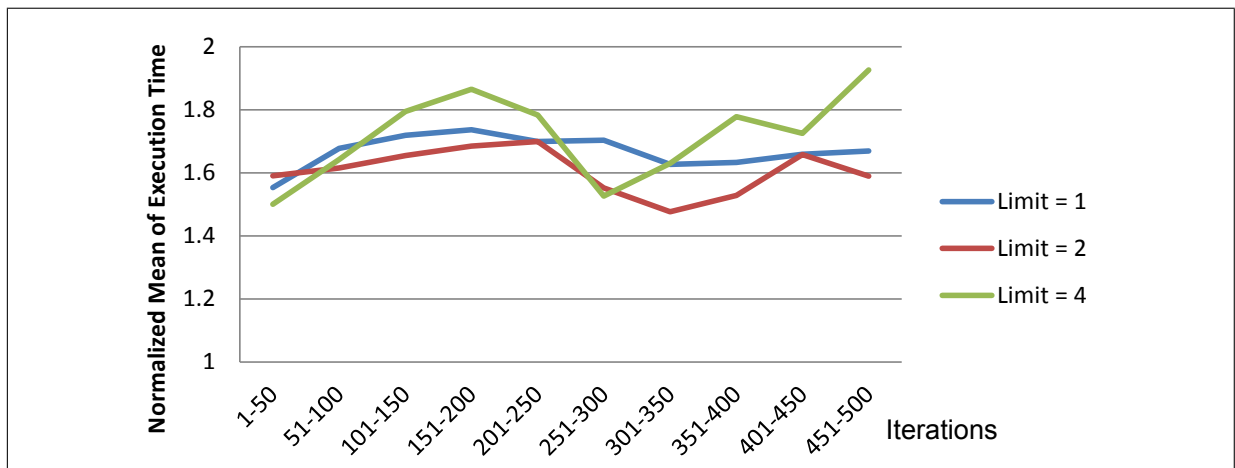


Figure 4.16: Execution times in LFCFS for workflow 1

4.4.5.4 Proportional Adaptive Task Schedule

From Table 4.2 we can see that the Proportional Adaptive Task Schedule gives the best cost efficient schedule and the second best execution times. Figure 4.21 shows the three workflows execution times when scheduled using this algorithm. We can conclude that this algorithm is the most cost-efficient and provides the optimal workflows makespan among the four algorithms we presented. It is 23.28% cheaper than the UFCFS, 19.74% cheaper than the best LFCFS variation and 25.61% cheaper than the best Pool-based Adaptive Task Schedule variation. Additionally, we notice that the Proportional Adaptive Task Schedule algorithm is the most efficient from a resource utilisation point of view (almost twice as efficient as the UFCFS).

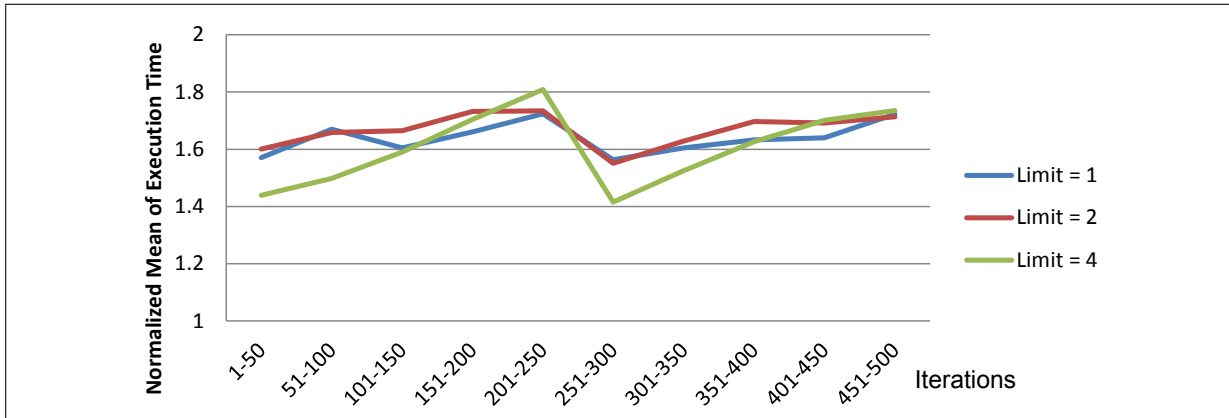


Figure 4.17: Execution times in LFCFS for workflow 2

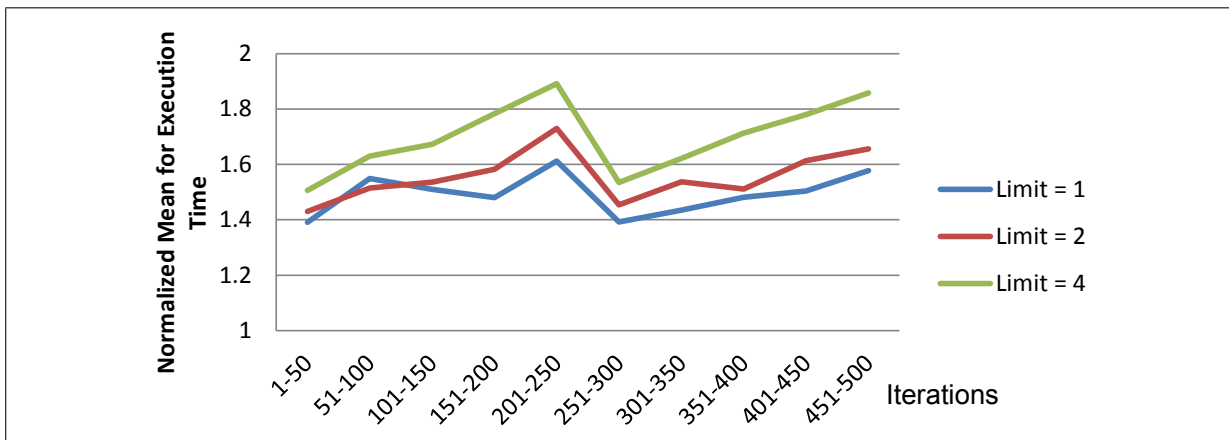


Figure 4.18: Execution times in LFCFS for workflow 3

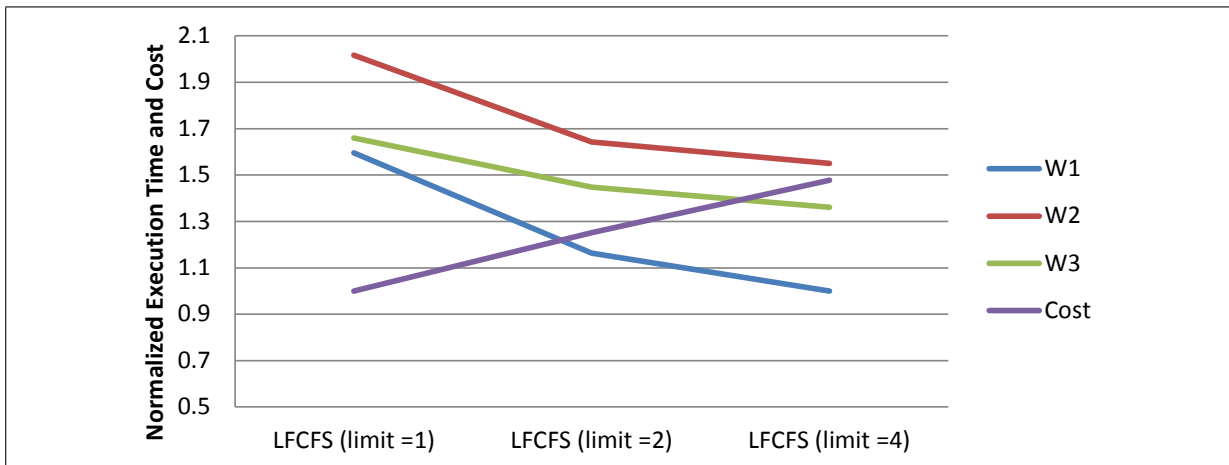


Figure 4.19: Execution time and cost benchmark in LFCFS

4.5 Summary

In this chapter, we have highlighted the need for cost-efficient scheduling of software process workflows in the cloud without causing significant delays in the execution time. We have shown that software process workflows contain different types of activities compared to scientific workflows and that the state-of-the-art scheduling approaches

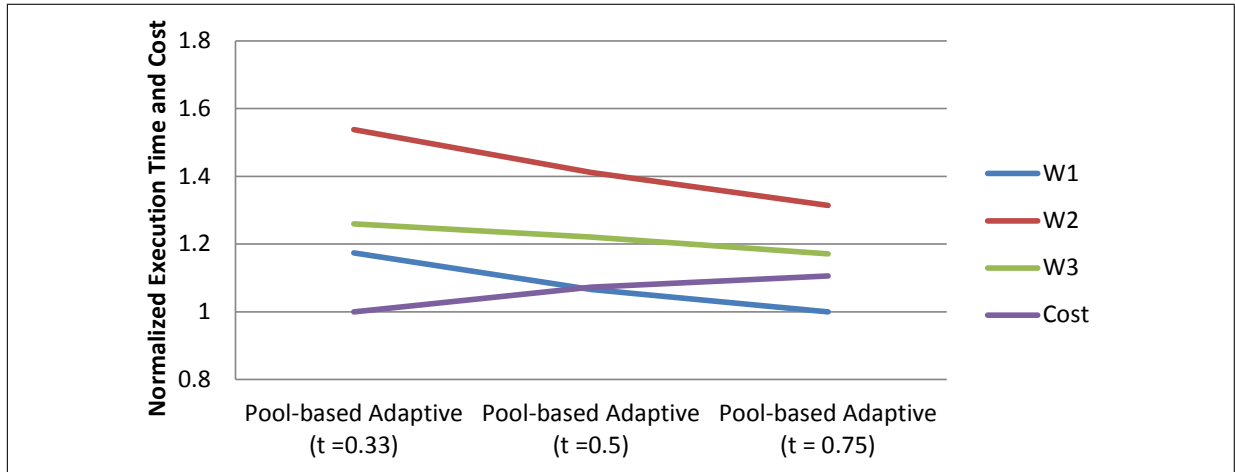


Figure 4.20: Execution times and cost in Pool-based Adaptive Task Schedule

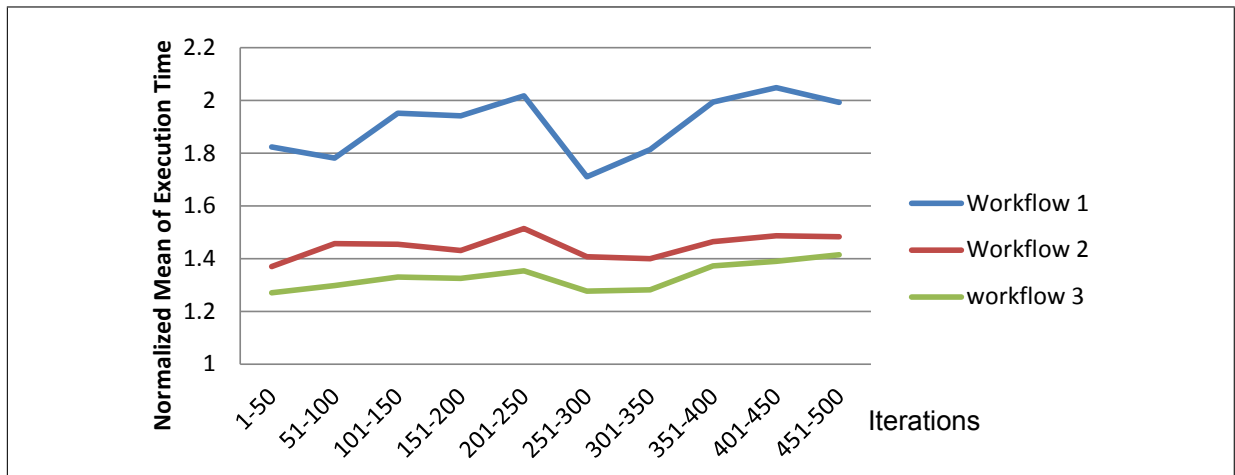


Figure 4.21: Execution times in Proportional Adaptive Task Schedule

do not meet the requirements of executing software process workflows. To meet these requirements, we adapted three algorithms; the Unlimited First Come First Serve (UFCFS), Limited First Come First Serve (LFCFS) and the Pool-based Adaptive Task Schedule. We also proposed a fourth one; the Proportional Adaptive Task Schedule. We evaluated their performance (through simulation) in terms of overall execution cost and execution times of individual workflow instances. The simulation results show that the UFCFS gives the shortest makespan while our proposed Proportional Adaptive Task Schedule gives the most cost-effective schedule, the best resource utilisation and the second best makespan. Unlike the LFCFS and the Pool-based Adaptive Task Schedule, the Proportional Adaptive Task Schedule does not rely on any arbitrary values and balances between the execution cost and time. The Proportional Adaptive Task Schedule algorithm is integrated in the proof-of-concept of the SDaaS reference architecture. In the next chapter, we report on evaluating the SDaaS architecture through a

case study where we use the proof-of-concept implementation to execute safety-related processes.

5

EVALUATION: A CASE STUDY ON CLOUD-BASED ENGINEERING OF SAFETY-CRITICAL SYSTEMS PROCESSES

Contents

5.1	Introduction	90
5.1.1	The evaluation method	90
5.1.2	The safety-critical systems case study	91
5.2	EXE-SPEM for Modelling Safety-related Processes	94
5.3	The PSSA Case Study	94
5.3.1	Argument generation	97
5.3.2	Implementation	101
5.3.3	Execution	103
5.4	Discussion	105
5.5	Summary	107

An earlier version of this chapter is published in: S Alajrami, B Gallina, I Sljivo, A Romanovsky, P Isberg: **Towards Cloud-based Enactment of Safety-related Processes**. In: Proceedings of the 35th International Conference on Computer Safety, Reliability and Security (SafeComp'16). Trondheim, Norway. September, 2016

5.1 Introduction

In previous chapters, we proposed the SDaaS reference architecture and the modelling language EXE-SPEM. In this chapter, we report on evaluating this cloud-based development approach by applying it in a case study from the safety-critical systems domain. In the next subsection, we reason about the evaluation criteria used in this chapter, while in Section 5.1.2, we provide the context of the case study used.

5.1.1 *The evaluation method*

Evaluating software development approaches using controlled experiments is challenging. This is because such approaches are applied in different contexts with multiple variables which are difficult to control [40]. Software architecture evaluations approaches such as SAAM [75], ATAM [76], SAAMCS [81], etc. focus on assessing if an architecture fulfils a set of requirements or not. We believe such approaches are suitable for evaluating commercial systems architectures but not for evaluating the SDaaS architecture we proposed in Chapter 2. This is mainly because these approaches require involving some/all related stakeholders in the evaluation process. In the SDaaS case, this would mean involving real developers and software project managers among other stakeholders which is not practical at this stage. We believe that the ultimate evaluation approach is to empirically evaluate the impact of using the SDaaS approach on developers (e.g., productivity), projects (e.g., cost, time and product quality). However, this is also not feasible at this stage given the timing constraints for this study. Therefore, we use a case study approach to evaluate the feasibility of the SDaaS vision.

Case studies have been used to empirically evaluate software engineering approaches [83, 97]. We instantiate the SDaaS reference architecture as we described in Chapter 2. We use the instantiated proof-of-concept to execute a safety-related process which we model using EXE-SPEM (see Chapter 3).

In Chapter 2, we argued how the SDaaS architecture meets a set of requirements. In this chapter, the case study aims to validate the claims listed in Table 5.1.

Table 5.1: Claims to be validated using the case study

No.	Claim
C1	Applicability. The SDaaS reference architecture is applicable to domain-specific processes (safety-critical processes in this case).
C2	Extensibility. Our software process modelling language; EXE-SPEM, is capable of modelling cloud-based executable processes and can be extended to fit domain-specific requirements.
C3	Openness. Command line tools and parts of the tools that has a GUI can be integrated in the SDaaS architecture as activities which can be used to construct process models.
C4	Provenance. The provenance data about process execution and the process models can be used to provide insightful knowledge.
C5	Automation. The SDaaS architecture supports automating parts of software processes and enables automating some originally non-automated activities.
C6	Potential. The SDaaS architecture can save cost and time spent on system development.

5.1.2 The safety-critical systems case study

Safety-critical systems engineering has to follow best practises. More specifically, safety standards (such as ISO 26262, ARP4761) provide guidance in terms of reference process models for the development and assessment of such systems. The complexity of such systems is reflected in their supply chain, which consists of a complex, geographically-distributed and heterogeneous supply network. Manufacturers rely on a number of suppliers, who are in charge of supplying software or hardware components needed for the assembly of the systems to be produced or for the automation of certain activities during the production. The reference processes recommended by the standards take into consideration the complexity of the systems and their supply network.

To be released on the market, the integrated systems must be certified. The certification process in various domains is conducted by scrutinising an argument supporting system safety [98]. In the automotive and rail domains, for instance, such argument is known as the *safety case*. In the aerospace domain, an explicit safety case is not required however as discussed by Holloway [69] an implicit safety case request is contained within the standards. Thus, all safety-critical systems must be accompanied by a safety case that provides assurance. There are two ways of providing assurance (that is building a safety case): by product and by process.

Safety cases can/should also reflect the compositional nature of the systems under

examination. Contract-based safety case fragments should be provided by suppliers and integrated within a complete safety case by the manufacturer, as the safety case structure proposed within EN50129 [3] in the rail domain might suggest. Even the provision of a safety case may follow a reference process [8].

The planning and execution of all the recommended reference processes is a time consuming and costly activity. Moreover, given the compositional and geographically distributed nature of the supply network, different interpretation of the processes may coexist resulting in conflicts and ultimately risk of low-quality products.

While the considerations listed in this chapter hold for several complex safety-critical systems, we focus on aircraft as an example of such systems. To engineer and certify an aircraft, a set of standards is at disposal to address various aspects such as safety assessment; system, software, and hardware engineering, etc. Typically, these standards provide requirements that should be followed to define the process to be used during the development and assessment of the aircraft and the software and hardware to be integrated within the aircraft. To define such process, a safety manager may refer to a reference model or may define a customised one by selecting and composing compliant process elements. To do the latter, the safety manager has to identify: the tasks to be executed in the correct order to consume/produce expected artefacts, roles, specific techniques to be used and in some cases the tools to automate the tasks. A document aimed at showing process compliance by providing a process-based argument is typically required.

Besides the process requirements, safety standards also include product requirements aimed at assessing the level of a product's safety based on the product's behaviour against the formulated safety requirements. Various analysis and verification results may be used to show that the product behaves as it should. Since we cannot guarantee that the final product is acceptably safe, standards are recommending to include a product-based argument to assure that the system is acceptably safe [61]. Additional requirements target the assessment process, which in many application domains is conducted by scrutinising an explicit or implicit safety case. A complete safety case as the final output of the assessment process should contain both the process and the product-based arguments to assure that the system development has not only followed the mandated process, but has also resulted in an acceptably safe product.

The case study presented in this chapter uses the Preliminary System Safety Assessment (PSSA) process from ARP4761 [2] as an example of safety-related processes. We model and execute this process in the cloud and use it to validate the claims mentioned in the Table 5.1.

Our vision is that a manufacturer models the planned safety life-cycle as well as the corresponding argumentation process. The process model enactment can be distributed geographically. The stringency (i.e., integrity level) with which the process tasks are performed is indicated via a standardised process modelling language (EXE-SPEM). By doing this, conflicting interpretations between teams can be reduced.

Using the cloud as an enactment platform not only reduces cost (through the pay-as-you-go and on-demand acquisition models), but also provides an accessible platform for the distributed teams involved in the system engineering process. Additionally, artefacts from across the different geographical locations can be maintained centrally which together with provenance data can facilitate the collection and processing of evidence supporting the system's safety case. Furthermore, the cloud's elasticity allows for acquiring more computational resources as needed for computationally intensive tasks.

The evaluation of the SDaaS architecture using this case study is achieved by:

- Instantiating the SDaaS reference architecture (see Chapter 2) and using the instantiated prototype to support engineering of safety critical systems.
- Implementing an activity to automate the generation of a fragment of a safety argument arguing about the safety characteristics of the produced system. The fragment is generated by analysing the results of the Failure Logic Analysis (FLA) of the system [54]. The analysis captures product-related evidence (e.g., detecting partial and full mitigators of failures).
- Implementing an activity to automate evidence capturing and automatic generation of process-related safety argument fragments. Process-related evidence include information about the process, stakeholders, tools and standards.
- Enacting an augmented PSSA process with automated safety argument fragments generation and presenting the generated argument fragments in visual, textual

and machine readable formats. These fragments can then be manually integrated within a complete safety case for the system.

5.2 EXE-SPEM for Modelling Safety-related Processes

In Chapter 2, we proposed the SDaaS architecture (a cloud-based architecture for software process enactment). The architecture adopts a model driven paradigm where processes are modelled and enacted. In Chapter 3, we have proposed the modelling language EXE-SPEM for modelling cloud-based executable processes. In this section, we will see how EXE-SPEM can be used to model safety processes which can be enacted in the SDaaS architecture.

EXE-SPEM focuses on modelling cloud-based executable process elements. We extend EXE-SPEM to enable capturing safety-related attributes for process activities. These attributes are: certification information for roles, the confidence of tools (supporting process activities) and the guidance and the standard each activity in the process adheres to. This information is used to support the process-based argument generation. As explained in Chapter 3, EXE-SPEM models are executable in the cloud. This is achieved by incorporating execution logic (order, preconditions, tools, versions, etc.) and cloud resource requirements (virtual machine image type, number of machines, etc.) into the model.

Each activity can be configured to use an initial set of computational resources and can also be set to automatically scale these resources after a predefined timeout. **The extensibility of the modelling language to fit different domains relates to claim C2: Extensibility in Table 5.1.**

5.3 The PSSA Case Study

PSSA examines the system architecture to identify how the system failures contribute to the failure conditions from these identified in the system Functional Hazard Assessment (FHA) (see Appendix C for details about PSSA and FHA). One of the tasks performed during PSSA is: determining if the system architecture and concept design can meet the safety requirements. In this case study, we focus on that portion of PSSA (as an

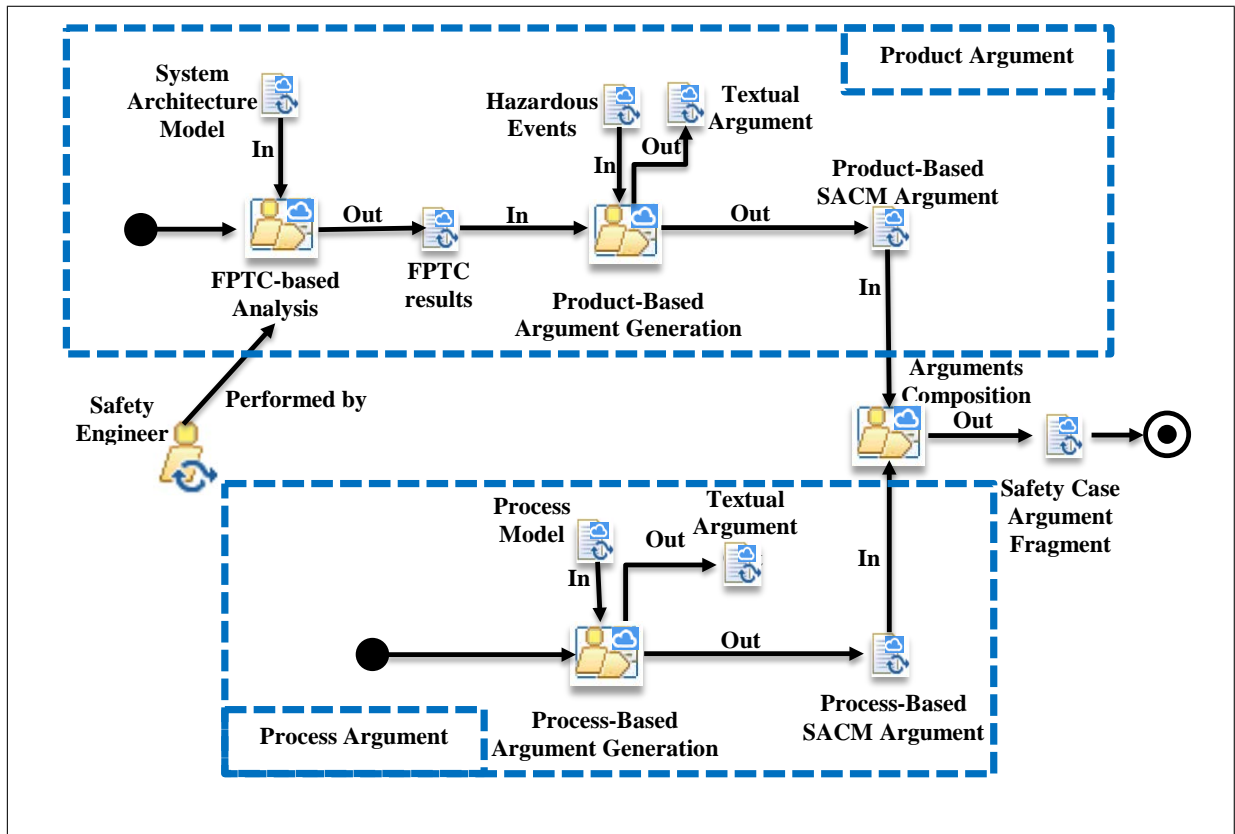


Figure 5.1: PSSA augmented with the argument generation process

example of a safety-related process) which is usually performed with the help of Fault Tree Analysis (FTA)(see Appendix C for details on FTA). In addition, we automate the generation of both product and process based argument fragments by analysing the FTA results (for the product argument) and analysing the process execution provenance data and the process model (for the process argument) as we explain in Section 5.3.1.

Figure 5.1 shows the EXE-SPEM model of the PSSA portion augmented with the argument generation process. It consists of the following four activities: FPTC-based Analysis, Product-based Argument Generation, Process-based Argument Generation and Arguments Composition. The *FPTC-based Analysis* activity uses Fault Propagation and Transformation Calculus (FPTC) to calculate the system level failure behaviour based on the failure behaviour of the individual system components (see Appendix D for more details about FPTC). It takes as an input the system architecture model and generates as an output the failure behaviour of the system. This failure behaviour can be used by the next activity (*Product-based Argument Generation*) to verify if the *undesired hazardous events* (identified after performing FHA) have been mitigated. The *Process-based Argument Generation* activity uses the process model and the SDaaS archi-

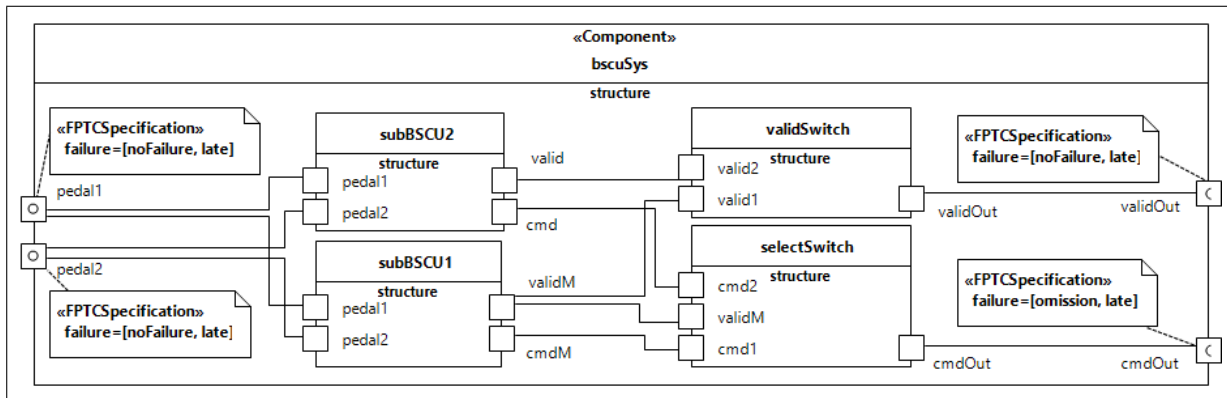


Figure 5.2: The architectural model of BSCU components, ports and failures [2]

texture provenance data to extract process and execution related information which can be populated as a process-based safety argument by following an argument pattern which we describe in Section 5.3.1. Finally, the *Arguments Composition* activity combines both the product and the process based arguments into one safety argument fragment. This fragment is then manually compiled with other fragments arguing about other safety aspects of the system (e.g., about other process portions from ARP4761) to form the product's safety case.

In this case study, we use the aircraft Wheel Brake System (WBS) from ARP4761 [2] as a well-cited example of a safety-critical system. The WBS is described in Appendix D. Here we will limit our attention to the portion of the WBS architecture that comprises the BSCU and its sub-components (as shown in Figure 5.2).

Since performing the FHA process for the WBS system is out of the scope of this case study, we make an assumption that the *undesired hazardous events (HEs)* are randomly selected. These hazardous events are the ones that the system should mitigate. A list of these hazardous events is provided as an input to the *Product-based Argument Generation* activity. Each undesired HE is accompanied by a definition of its criticality level. Criticality levels names vary across different standards. For instance, in ARP4754A [6], the levels are: *negligible*, *minor*, *major*, *hazardous*, *catastrophic*. To abstract this variance between standards, the levels are mapped to a five-level numerical criticality scale ranging from 1 (lowest criticality) to 5 (highest criticality).

To summarise, in this case study, we model a portion of a standardised process (PSSA from ARP4761) and introduce automation of parts of this process portion (for generating argument fragments) and execute the model in the SDaaS cloud-based architecture.

Not only the SDaaS architecture provides a central accessible platform for creating and storing the model and artefacts, but also enables the generation of the safety fragments and their relevant evidence (as explained in the next subsection). Section 5.3.2 reports on the implementation of the individual activities used in the PSSA model presented here while Section 5.3.3 describes the execution of the model in the SDaaS prototype and the generated outputs.

5.3.1 Argument generation

The manual creation of a safety case is a costly and time-consuming process, thus automation can be used to generate parts of the process and product-based arguments. The generated argument fragments can be then composed to obtain the complete safety case, with the possibility of customised tailoring of the links between the generated fragments. The two prerequisites for the argument fragments generation are: a) the source of information for the content of the arguments; and b) the target argument structure and format.

The SDaaS architecture captures provenance data related to the safety process and stores the artefacts used and produced during the process. The benefit of such data and artefacts is that it can be the foundation for safety cases. However, manual extraction of safety cases from this raw data is an expensive, time-consuming and error prone task. Consequently, we take a step further and automate the generation of safety argument fragments arguing about both the product and the process aspects of safety.

5.3.1.1 Product-based argument

The product-based argument aims at showing that the product behaves as it should. To automate the generation of such argument, the analysis and verification results can be exploited. We build on top of previous study [102] and we extract information about the failure behaviour of the system from the FPTC analysis results.

FPTC analysis results calculate the failure behaviour of a safety-critical system (see Appendix D for details about FPTC). Further analysis of these results can determine if certain failures/hazardous events (*HEs*) occur or not. This allows us to argue about how the system handles HEs. If an HE is present in the system, we produce a counter-evidence in the form of a trace to the source(s) of the HE. If it is not, we find the

component(s) that mitigated it. Mitigation can be partial or full. Full mitigation is when the failure does not propagate from a component's input to its output while partial mitigation is when the failure is present on the output, but at least one of the input causes of the output failure has been mitigated by the component. The analysis for product-based argument fragment generation starts by parsing the FPTC results and following the pseudo code in Figure 5.3. Then the argument is formulated by constructing *Claims* and *Strategies* and supporting them by *Evidences/Counter-Evidences* following the rules in Figure 5.4. These rules are adapted from [102] where arguments were generated from safety contracts. Representations of safety arguments are discussed in Appendix E.

```

S: the set of system components;
HE: the set of undesired hazardous events
M: list of mitigators;
PM: list of partial mitigators
for each he in HE
{
    if(he.criticality > negligible)
        if(he exists on the system output)
            trace_failure_to_the_source();
        else
            for each component s in S
                if(he is present on s.input)
                    if(he is not on s.output){
                        M.add(s);
                        find_the_mitigating_rule(); }
                else
                    if(the source of he on s.output != s.input)
                        PM.add(s);
}

```

Figure 5.3: The pseudo code for analysing the FPTC results

5.3.1.2 Process-based argument

The process-based argument fragment aims at showing that the process mandated by the corresponding standard has been followed. The MDSafeCer (Model-driven Safety Certification) method [56] can be used to automate the generation of such arguments. Via MDSafeCer, process models compliant with e.g., SPEM2.0 are transformed into composable process-based argumentation models compliant with e.g., SACM and presented via e.g., GSN goal structures (see Appendix E for details about safety cases


```

R1: Make CLAIM "All causes of hazardous Failure Modes are acceptable"
R2: For each hazardous event {he} in the set HE, apply the following:
  R2.1: If {he} is negligible, make a CLAIM "Hazardous Failure Mode {he}
        is negligible"
  R2.2: If {he} is not negligible, make a CLAIM "Hazardous Failure Mode
        of type {he} absent in contributory software functionality" and
        attach CONTEXT "Known causes of {he} failure mode"
    R2.2.1: If {he} is present on the output, make COUNTER-EVIDENCE "The
            {he} Hazardous Failure Mode present in the contributory software
            functionality. Check traces."
    R2.2.2: If {he} is not present on the system output, make a STRATEGY
            "Argument over failure mechanisms" and attach a JUSTIFICATION "
            Identified failure mechanisms describe all known causes of {he}
            hazardous Failure Mode"
      R2.2.2.1: make a CLAIM "The known causes of secondary failures of
                    other components are acceptably handled" and leave it
                    undeveloped.
      R2.2.2.2: make a CLAIM about the mitigators "Hazardous event {he}
                    has been mitigated by {mitigators}" and attach an EVIDENCE "
                    Mitigation details in the textual argument"

```

Figure 5.4: Rules for product-based argument construction

representation). This method supports compositional argumentation and reuse. Some of the aspects that should be covered in such an argument are the tools and techniques used as well as the qualification of both the tools and the persons using those tools and techniques. A model of such a process is needed as the source of information for the process-based argument generation. SPEM2.0 is a modelling language that can be used to model such a process, which can then be used as the source model for generation of the target process argument fragments [56]. Therefore, we can use our extended version of SPEM2.0 models; EXE-SPEM models as a source model for generating process-based argument fragments.

MDSafeCer provides rules for mapping a subset of SPEM2.0 elements into GSN and SACM concepts [57]. These rules are shown in Table 5.2. Using these rules, process model elements (expressed in SPEM2.0) can be mapped into a safety argument represented in either GSN or SACM.

MDSafeCer also presents a set of rules for structuring the process-related safety argument. It starts with a top claim arguing that the process has been compliant with the standards. This claim is then decomposed further until it reaches an atomic process-

Table 5.2: Concept mapping as proposed by MDSafeCer [56]

SPEM2.0	GSN	SACM
Task ta	Goal	Claim
Role ro	Solution	InformationElement
Work product wp	Solution	InformationElement
Tool to	Solution	InformationElement
Guidance gu	Solution	InformationElement
Relationship between ta and $ro/to/wp/gu$	supportedby	AssertedEvidence

related unit [57]. The detailed rules for structuring a GSN process-based safety argument fragment [57] are shown below. These rules are applied for each activity in the process model.

1. Create the top-level goal ID:G1 and statement: “The task ta has been carried out”. Create the context to be associated to G1. Context ID:C1 and statement: “Standard x ”, where x is a variable. Create an inContextOf link to relate G1 and C1. Develop the goal G1 further by creating four strategies and for each strategy a set of sub-goals.
 - (a) S1: “Argument over roles R ”.
 - (b) S2: “Argument over work products W ”.
 - (c) S3: “Argument over tools T ”.
 - (d) S4: “Argument over guidance G ”.
2. Further develop strategy S1 and for every role ro in R : create a goal G1.ro “ ro is certified” and develop this goal further by creating the corresponding solution E.ro “ ro ’s certifications” and the supportedBy links necessary to link S1 with G1.ro and G1.ro with E.ro.
3. Further develop strategy S2 and for every work product wp in W : create a goal G1.wp “ wp is available” and develop this goal further by creating the corresponding solution E.wp “ wp -related name” and the supportedBy links necessary to link S2 with G1.wp and G1.wp with E.wp.
4. Further develop strategy S3 and for every tool to in T : create a goal G1.to “ to is qualified” and develop this goal further by creating the corresponding solution E.to “ to ’s qualifications” and the supportedBy links necessary to link S3 with G1.to and G1.to with E.to.

5. Further develop strategy $S4$ and for every guidance gu in G : create a goal $G1.gu$ "Guidance gu has been followed" and develop this goal further by creating the corresponding solution $E.gu$ " gu where and how" and the supportedBy links necessary to link $S4$ with $G1.gu$ and $G1.gu$ with $E.gu$.

5.3.2 Implementation

After modelling the PSSA augmented with the argument generation process (Figure 5.1), Model to Text transformation is applied on the EXE-SPEM as prescribed in Chapter 3. The resulted XML model is then enacted in the SDaaS architecture prototype. Below, the implementation of each of the activities used in the augmented PSSA process is detailed.

- **FPTC-based analysis**

As mentioned in Section 5.3.1, the FPTC analysis is conducted to discover the fault propagation behaviour in the system. This activity uses Concerto-FLA (the extended FPTC implementation from the CONCERTO project ¹) to perform the FPTC analysis. The CONCERTO tool-set allows: creating UML-based architectural models of the system; performing FPTC analysis (using Concerto-FLA) including back-propagation of the results visually on the models. The architectural model is transformed to the *flamm* format (an XML-like format) on which the analysis takes place. The *flamm* model consists of composite components (systems) containing atomic components. The (atomic) components have input and output ports where failures are attached. In addition, each component has a set of rules defining its failure behaviour. For this case study, we have extracted the FPTC analysis part from Concerto-FLA into this standalone activity which generates a *flamm* model including the analysed failure behaviour of the system.

- **Product-based argument generation**

This activity uses the FPTC analysis results to construct the argument concerning the BSCU. The FPTC results are embedded in the output *flamm* model which makes it very hard to be extracted by hand. Therefore, this activity parses the output *flamm* model and, as described in Section 5.3.1, looks for the non-negligible

¹www.concerto-project.org/

undesired hazardous events in it. Once a non-negligible *undesired hazardous event* has been found, it is traced down to its causal source (if it is present on the system output). If the undesired hazardous event is not present on the system output, the mitigating component which prevented it from propagating to the system output is identified. Based on this analysis, an argument arguing whether the system acceptably handles a set of hazardous events (**HE**) or not is constructed following the rules described in Figure 5.4. This argument is supported with evidences/counter-evidences. The constructed argument is represented in two formats: a) a machine readable format (SACM/XMI) which can be visualised into a GSN argument using external tools such as Astah GSN editor ², and b) a textual format using the Argument Outline which is described in Appendix E. While the SACM and GSN arguments do not contain the detailed traces of failure propagation and mitigation information for brevity, they refer to the textual argument which contains these information.

- **Process-based argument generation**

This activity follows the rules described in Section 5.3.1 for constructing a process-based argument fragment arguing about the compliance of the process with the chosen standard/practice (PSSA from ARP4761 in this case). For each activity in the process, a portion of the argument is constructed by extracting information such as the guidance followed, the confidence of the tool, etc. Similar to the product-based argument, this argument is generated in both SACM/XMI (externally visualised into GSN) and argument outline textual formats.

- **Arguments composition**

Once the product and process based arguments are generated, this activity takes as an input both arguments in the SACM/XMI format and combines them into a single argument arguing about both aspects of the system safety. This is done by adding a top *Claim* (Goal) arguing about the safety of the system overall. The output of this activity is again presented in both SACM/XMI and argument outline textual format.

²<http://astah.net/editions/gsn>

The tools described above (whether extracted from existing tools or created from scratch) relate to claim C3: Openness in Table 5.1.

5.3.3 Execution

The process model shown in Figure 5.1 is executed in the SDaaS prototype we instantiated in Chapter 2. We deployed the *Enactment Service* and one *Workflow Engine* on two different Amazon EC2 "t2.small" machines. Using a web browser, we were able to execute the process and retrieve the generated artefacts containing the FPTC analysis results and the safety arguments (separate and combined) in both SACM/XMI and text formats (see Appendix E for argument representation formats). The SACM/XMI formats were then converted into GSN diagrams using the Astah GSN editor. Here we detail what each activity in the process consumed and produced.

FPTC-based analysis

Input: the textual representation of architectural model of the BSCU component as illustrated in Figure 5.2. The textual representation is in the *flamm* format. The full model can be found in Appendix F.

Output: the *flamm* model enriched with the failures found on each component output ports. The output model can be found in Appendix G.

Product-based argument generation

Input: the FPTC analysis results as shown in Appendix G and a list of *undesired Hazardous Events (HEs)* which can be found in Appendix H. Here we assume that the hazards: *omission* and *late* are identified in the FHA process as undesired hazards and have a criticality level of 5 and 2 respectively.

Output: the product-based argument fragment represented as both SACM and textual arguments. The SACM argument (can be found in Appendix I) is visualised into the GSN graph in Figure 5.5. A snippet of the textual representation is shown in Figure 5.6. The full textual representation containing all the traces and mitigation information can be found in Appendix J. **The automation of this argument generation relates to claim C5: Automation in Table 5.1.**

Process-based argument generation

Input: the process model which can be found in Appendix M.

Output: a process-based argument fragment represented in both SACM and textual

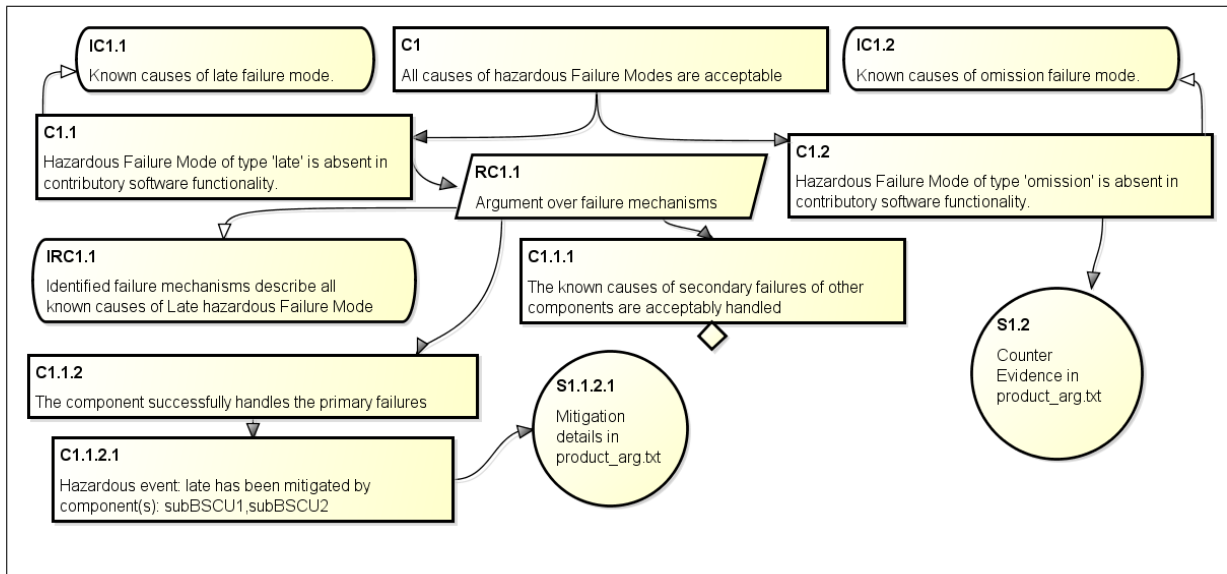


Figure 5.5: GSN representation of the generated product-based argument

```

...
CLAIM 1.1: HAZARDOUS FAILURE MODE OF TYPE 'OMISSION' IS
ABSENT IN CONTRIBUTORY SOFTWARE FUNCTIONALITY.
CONTEXT 1.1: Known causes of omission failure mode.
COUNTER_EVIDENCE 1.1: The omission Hazardous Failure
Mode is present in the contributory software
functionality. Check the traces.
CONTEXT 1.1: omission CAUSED BY:
Failure: 'omission' On Output Port: 'cmd' of
Component: 'selectSwitch'.
CAUSED BY: {Failure: 'omission' On Input Port:'cmd2'
of Component: 'selectSwitch'.
CAUSED BY: Failure: 'omission' On Output Port:'cmd'
of Component: 'subBSCU2'.
CAUSED BY:
...

```

Figure 5.6: The product-based argument represented in text

representations. The SACM is visualised into GSN and a portion of the GSN graph showing the part of the argument about the FPTC-based Analysis activity is shown in Figure 5.7. The respective portion of the textual argument is shown in Figure 5.8. The full SACM and textual process-based argument can be found in Appendices K and L. **The generation of the process argument generation using provenance data relates to claim C4: Provenance in Table 5.1.**

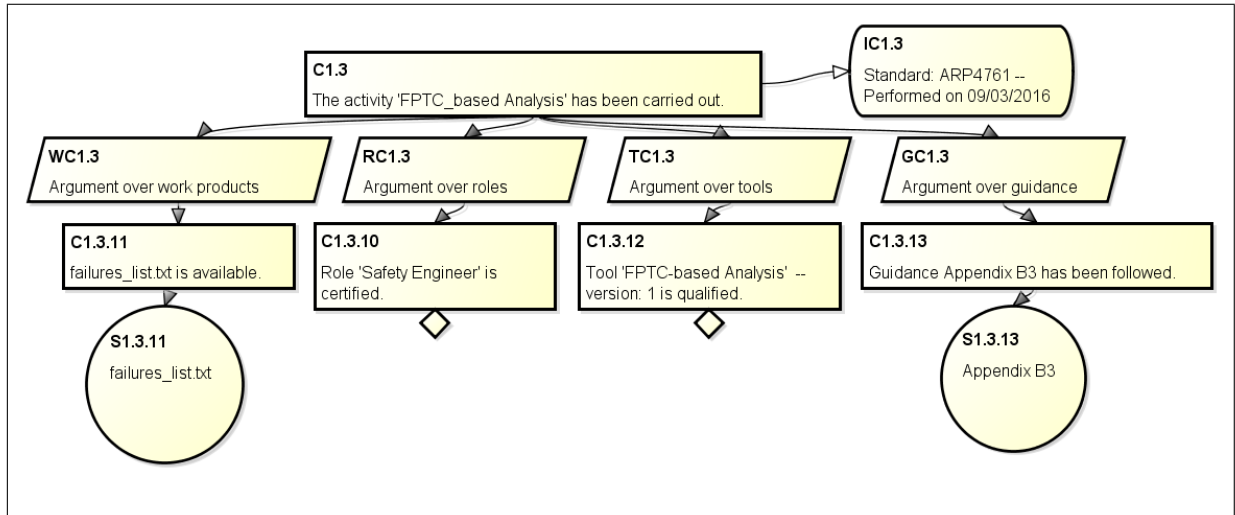


Figure 5.7: GSN representation of partial process-based argument

```

...
CLAIM C1.3: THE ACTIVITY FPTC_based Analysis HAS BEEN CARRIED
OUT.
    CONTEXT C1.3: Standard: ARP4761 -- Performed @
09/03/2016
    STRATEGY RC1.3: Argument over roles
        CLAIM C1.3.10: ROLE SAFETY ENGINEER IS
CERTIFIED.
            [Undeveloped]
    STRATEGY WC1.3: Argument over work products
        CLAIM C1.3.11: FPTC results IS
AVAILABLE.
            EVIDENCE S1.3.11: bscu.flamm
    STRATEGY TC1.3: Argument over tools
        CLAIM C1.3.12: TOOL FPTC_based Analysis
-VERSION: 1 IS QUALIFIED.
            [Undeveloped]
    STRATEGY GC1.3: Argument over guidance
        CLAIM C1.3.13: GUIDANCE Appendix B3 HAS
BEEN FOLLOWED.
            EVIDENCE S1.3.13: Appendix B3
...

```

Figure 5.8: A portion of the process-based argument represented in text

5.4 Discussion

In Table 5.1, we have listed a set of claims to be validated about the SDaaS architecture. In this section, we will discuss how the case study presented in this chapter has validated those claims.

- **C1: Applicability.** *The SDaaS reference architecture and approach is applicable to domain-specific processes (safety-critical processes in this case).*

While we have used a process from an aerospace domain standard, processes from other standards can be modelled and enacted similarly. Although some other domains might need specific requirements, it is possible to extend the EXE-SPEM modelling language to include such requirements in the process model and extend the architecture to handle these requirements if necessary. Alternatively, new activities can be created and used in the process models to deal with such requirements. We have used the latter approach when we created the process-based argument generation activity. This activity used the safety-related information in the process model to generate a process-based safety argument.

- **C2: Extensibility.** *Our software process modelling language; EXE-SPEM, is extensible and can be extended to fit domain-specific requirements.*

As mentioned above, we have embedded safety-related elements (e.g., role certification and tool qualification) in EXE-SPEM models. Similarly, additional domain specific elements could be added to EXE-SPEM meta-model to support other domains which require specific modelling elements.

- **C3: Openness.** *Command line tools and parts of the tools that has a GUI can be integrated in the SDaaS architecture as activities which can be used to construct processes.*

In the case study, we have developed four different activities. The FPTC-based analysis activity was extracted from the Concerto tool-set which is an eclipse-based platform. We extracted the core functionality of the FPTC analysis from this platform and used it as a standalone activity. The other three activities were developed from scratch. In Chapter 1, we have wrapped command line tools (e.g., Spin [59] and DiVinE [23]) as activities and used them to run our initial experiments. Therefore, different types of tools can be integrated in the SDaaS architecture. However, the heavily interactive GUI tools cannot be supported as they are, and would require their logic to be implemented as a standalone tool.

- **C4: Provenance.** *The provenance data about process execution and the process models can be used to provide insightful knowledge.*

In the case study, the process-based argument generation activity used the process model and provenance data (e.g., which actor performed certain activities) to build the process-based safety argument. This argument is an insightful evidence documenting the process that has been followed. Without the provenance data, such evidence would have had to be constructed manually.

- **C5: Automation.** *The SDaaS architecture supports automating parts of software processes and enables automating some originally non-automated activities.*

The construction of safety cases is done by human experts who go through design, documentations and other artefacts to collect and scrutinise the evidences required to support their claims. As we have seen in the case study, we were able (thanks to the SDaaS architecture) to automatically generate fragments of the safety case.

- **C6: Potential.** *The SDaaS architecture can save cost and time spent on system development.*

Building on the previous point, human experts who construct safety cases are expensive assets. Automating significant part of the safety case construction saves significant amount of time and money needed for building those parts manually.

Based on the case study presented in this chapter, we can claim that the SDaaS architecture and approach is feasible to apply in different domains and in many scenarios provided that the required tools can be repackaged as activities from the existing tools or are built from scratch.

5.5 Summary

In this chapter, we have evaluated the SDaaS reference architecture using a case study from the safety-critical systems domain. We used the instantiated proof-of-concept

implementation of the architecture to execute a process model representing a portion of the PSSA process from the ARP4761 standard. The SDaaS architecture allowed us to execute the process model, automate parts of the execution and automatically generate safety argument fragments which can be used for safety cases. We also discussed how the case study validated a set of claims about the the SDaaS architecture. The next chapter concludes this thesis.

6

CONCLUSIONS

Contents

6.1	This Thesis in a Nutshell	110
6.2	Future Work	113
6.2.1	Motivating Scenarios	115
6.3	Concluding Remarks	116

6.1 This Thesis in a Nutshell

This thesis investigates undertaking software development in the cloud. More specifically, providing an alternative cloud-based development platform which can support the entire development life-cycle. This was motivated by the evolution of cloud computing which led it to be the enabling platform for software delivery in the Post-PC era.

Historically, software engineering has been evolving to exploit disrupting technologies and cope with management challenges. In the cloud computing (as a disrupting technology) context, the question becomes: *How to adapt software engineering for/in the cloud?*. While some research has investigated the software engineering practises **for** developing cloud applications (e.g., [100]), this thesis is the *first* study investigating a comprehensive approach for supporting software development **in** the cloud.

The main contribution of this thesis is proposing a reference architecture for supporting Software Development as a Service (SDaaS) in the cloud. This architecture uses the cloud economies of scale to provide computational resources, store software artefacts and provide an accessible development platform. It adopts a model-driven approach where software processes are modelled and transformed into executable workflow models. The use of software process models provides different levels of abstraction for different stakeholders and enables visualising, monitoring and tracking software development. Such models are mapped into a machine-readable format, then executed (enacted) in a distributed set of workflow engines. The workflow engines have different computational specifications (e.g., privacy and power) which allows the SDaaS architecture to cater for different software workflow activities. The architecture can scale by adding/releasing workflow engines on demand.

To materialise the SDaaS vision, this thesis investigates and contributes to the three following areas:

Modelling of cloud-based executable software processes

We found that the existing software process modelling languages do not support modelling cloud-related execution configurations. Many of them do not have native support for process enactment/execution. Therefore, we propose EXE-SPEM which is an extension of the OMG SPEM2.0 standard. EXE-SPEM is the *first* software process modelling

language for modelling cloud-based executable software processes.

Additionally, we provide an XML schema for representing the process models in a machine-readable format. We also provide rules for mapping EXE-SPEM models to this format.

EXE-SPEM inherits its extensibility from SPEM2.0. Domain specific elements can be integrated in the meta-model. We demonstrated this by integrating some safety-related attributes to EXE-SPEM meta-model for the case study we described in Chapter 5. Similarly, it can be done for other domains.

Even though interactions among stakeholders and between stakeholders and the process activities are valuable information which can be used to reason about the process, currently, EXE-SPEM only supports basic interactions between stakeholders and activities.

Cost-efficient scheduling of software processes execution in the cloud

Cloud offers a massive pool of resources on demand. While the cloud resources may seem deceptively unlimited, monetary resources are always limited. Software production cost is an important factor in the success/failure of software projects. Therefore, we believe that the SDaaS architecture should use cloud resources in a cost-efficient way. This inevitably means that a compromise on workflow makespan is necessary. However, we aim to minimise that compromise.

We propose the Proportional Adaptive Task Schedule algorithm which dynamically restricts the number of workflow engines (VMs) the SDaaS architecture can acquire within each operational hour. The maximum number is dynamically calculated periodically based on the proportion between the execution times of the expected activities to arrive in the next hour, and the execution times of the activities which started execution in the past hour. We evaluate the algorithm through simulation and by benchmarking it against three other adapted algorithms. The evaluation shows that our algorithm saves between 19.74% and 45.78% of the execution cost, provides best resource (VM) utilisation and provides second best workflows makespan compared to the other adapted algorithms. This algorithm is the *first* that targets scheduling software workflows in the cloud.

This algorithm can reduce the software production cost when used in the SDaaS archi-

ecture. However, as a result of this saving, the workflow makespan is increased. While process authors can statically define certain activities as priority activities (i.e., must be executed immediately regardless of the cost), the current approach does not allow the process author (or the stakeholder executing the process) to dynamically make a trade-off between execution time and cost. Also, it does not automatically make this trade-off on behalf of the user (e.g., by assigning weights to both cost and makespan and use them to calculate an optimal schedule). This is left as a future work.

Evaluating the SDaaS architecture

Evaluating the SDaaS architecture is a challenging task. As we discussed in Chapter 5, we evaluate the reference architecture by instantiating it and using its instance to conduct a case study from the safety-critical systems domain. We model a portion of a safety-related process from the aerospace domain and execute it in the cloud. Not only we were able to execute the process, but also -thanks to the process model and the SDaaS architecture provenance data- we were able to automatically generate fragments of a safety case arguing about both product and process-based safety aspects. We also demonstrated implementing workflow activities (development tools) on the cloud. Some of these tools were wrapped as a workflow activity while others were created from scratch. The evaluation proves the feasibility of the approach and the possibility to utilise it for different domains.

Limitations

The SDaaS architecture can be deployed into any cloud deployment model (public, private or hybrid). It can also be interfaced with existing platforms using service calls. This flexibility can address security and privacy concerns when using the cloud, i.e., one can use a private cloud to host the process enactment (partially or fully as each activity can be configured differently) and the generated artefacts.

However, there are some limitations to the type of activities that can be supported at this point. Software processes are often long-living and typically would involve human-intensive activities. The instantiated prototype of the architecture does not yet support intensive interactions with actors (humans) during process execution. Capturing those interactions provides more data which can be used to gain valuable insights about the process. Furthermore, a failure/exception during a long-running process will break the execution and the process will need to be restarted. It is essential to have

support to pause/resume processes in such situations. Since we do not have support to resume process execution in the case of failures, we recommend splitting long-living processes into short-living sub-processes. Sub-processing also means better separation of concerns between teams. Finally, not all activities within a process can be automated and the borders between what can/cannot be automated are not easy to define. The benefits from automation remain, however. For example, the automation of arguments generation in the case study, presented in Chapter 5, saves time and money.

6.2 Future Work

Throughout this thesis, we touched on some topics without going into details. Such topics include: SLA monitoring, real-time consistency checking, human-to-human interactions in software processes, mining software repositories and empirical evaluations. Each of these topics can be investigated in-depth in separate studies. In this section, we highlight some potential future directions to complement the work presented in this thesis.

Tool support for the SDaaS architecture

Implementation of tool support is essential for the adoption of the SDaaS architecture. The required tool support includes: tools for modelling and designing processes, full instantiation of the reference architecture supporting all types of process activities, and integration of more development tools. In addition, a tool discovery/catalogue service can be implemented to provide information about tools, guidance on how to use them and trade-offs between different tools.

Empirical studies

In this thesis, we evaluate the SDaaS architecture through a case study. While this validates the feasibility and applicability of the architecture, it does not evaluate the impact it would have on stakeholders (e.g., productivity and error rate) and projects (e.g., cost, quality and time). Therefore, an empirical study investigating those aspects is needed to analyse the impacts of using the SDaaS architecture in real software projects. Another aspect that needs to be empirically studied is the usability and accessibility of the architecture and how it will impact collaboration between (and across) teams.

On demand Micro-Tools

As Clark predicted in 1999 ¹, the world is becoming a network of services. Multiple development tools and environments already offer integration with other tools and platforms. In this thesis, we focus on modelling software processes as workflows and executing them in the cloud. The workflow building blocks (the activities) are either built-in beforehand or custom-made for a specific purpose. We demonstrated the creation of new activities in the case study presented in Chapter 5. Similarly, we believe tools could be built from aggregating *Micro-Tools* in a workflow which can then be executed in the cloud. Such *Micro-Tools* can be offered on demand and on a pay-as-you-go basis. This might even lead to a pay-per-feature-use model where you custom build your tool by choosing compatible *Micro-Tools* supporting a set of required features. We envision the granularity of *Micro-Tools* to be of atomic features (e.g., save a file, compile code, etc.) and that a *Micro-Tools* discovery/catalogue service can help choose the right *Micro-Tools* for building your custom tool.

Interaction patterns

As mentioned in Chapter 1, the increasing mobility of the Post-PC era has influenced the type of devices that are in use. Low specification, lightweight, mobile devices are becoming essential part in our everyday life. Projects such as *TouchDevelop* [21] exploit such devices for software development. New software development interaction patterns should build on the capacities of mobile devices (e.g., voice recognition and touch screens). Another aspect of interaction which can be studied is how stakeholders interact with workflow activities and among themselves (offline).

Big data for software development

In the SDaaS architecture, the processes are modelled, the artefacts are maintained (with different versions), stakeholders actions are recorded and provenance data about the process execution is collected. This data can be used for real-time and historical analysis. Such data can be utilised for the components of the architecture (see Figure 2.3) that we did not explore in depth in this thesis. These areas are: *SLA monitoring* when two or more organisations are collaborating, and *consistency checking*, to raise an alarm when processes divert from certain standard processes or constraints.

¹<http://www.nytimes.com/1999/04/18/business/economic-view-is-mr-gates-pouring-fuel-on-his-rivals-fire.html>

6.2.1 *Motivating Scenarios*

In this subsection, we briefly describe two scenarios which show the impact of the SDaaS architecture and the potential of the future work directions suggested in the previous section.

6.2.1.1 **Continuous delivery**

Continuous Delivery [70] has become a trendy software development paradigm. It aims at automating the build-test-deploy-release cycle. The motivation is to achieve frequent releases, reduce conflicts and therefore, reduce cost. To achieve such automation, teams should follow certain practises and use supporting tools/platforms. Humble and Farley [70] set the principles and technical practises for successful implementation of Continuous Delivery. An example of a continuous delivery process is the Facebook deployment pipeline [49].

Discussion

Systems like Facebook are delivered through the Internet where changes and new features are continuously pushed to users transparently. Faster and frequent releases (as prescribed in Continuous Delivery) mean that developers will be committing and releasing code very often (sometimes on daily basis). The benefits of such frequency are evident. Small and frequent releases mean easier bug locating and fixing as bugs will be in the newly added code which is small in size [70]. In addition, the code base is always maintained to be bug-free after each release which leads to reducing the required integration effort. Automation and repeatability of the software build-test-deployment-release are a key enabling factor for Continuous Delivery [70]. To pick up the fruits of Continuous Delivery, the management aspect of the development process must be considered. For example, if developers do not commit their code regularly, the Continuous Delivery chain is broken. Therefore, there is a need for convergence and monitoring support to ensure certain processes and practises are followed. SDaaS uses process models which can prescribe the recommended practice. Provenance data and consistency checking can ensure the required convergence. The cloud infrastructure provides the required tools on demand and also supports the automation of parts of the process.

6.2.1.2 Compliance and continuous certification

Small connected devices are being embedded everywhere from the human body to civil infrastructure and military applications. This means that more and more software systems are becoming safety-critical. Safety-critical systems must comply with certain regulatory standards and be certified by relevant authorities (as we discussed in Chapter 5).

Discussion

Many software components need to comply with certain domain-specific standards and regulations. This raises two important requirements for developing such components. First, the development team(s) must ensure their compliance with the adopted standard. Second, the development team must collect and retain evidence that they did so in order to build their case for certification. As Fuggetta and Di Nitto [53] state, the software community is challenged with the need to move from rigid compliance to smart convergence. This is especially important since in a human-centric process (like software development) it is impossible to force rigid processes and patterns. Implementing the SDaaS consistency checking component and defining consistency rules can help achieving such smart convergence with the help of the SDaaS provenance data and process models.

6.3 Concluding Remarks

In the lack of closely-related research on software engineering in the cloud, this thesis presents the *first* step towards achieving a transition from the desktop-based development environments to cloud-based environments and interconnected tools. Such a transition is inevitably happening. However, academia has been lagging behind industry in investigating this area. Industrial vendors are moving their development tools or creating new ones in the cloud without moving towards a comprehensive vision for Software Development as a Service.

This thesis does not only come to fill an existing gap or meet some needs, but it also paves the way for future opportunities and provides a high level road map for further research in the area of software engineering in the cloud as we have shown in the Section 6.2. Despite the potential of this approach, challenges in software development

will continue to exist. Indeed, as Fred Brooks puts it, "*There is no silver bullet*" and we can only eliminate accidental difficulties in software development. Inherent difficulties will continue to exist as software and its development evolve [30].

BIBLIOGRAPHY

- [1] *AC 25.1309-1A - System Design and Analysis*. Federal Aviation Administration (FAA), USA, June 1988.
- [2] *ARP4761: Guidelines and Methods for Conducting the Safety Assessment process on Civil Airborne Systems And Equipment*. SAE International, Warrendale, PA, USA, 1996.
- [3] *BS EN50129: Railway applications. Communication, signalling and processing systems. Safety related electronic systems for signalling*. Number BS EN 50129:2003. BSI Group, London, United Kingdom, 2003.
- [4] *Software Engineering – Metamodel for Development Methodologies*. Number ISO/IEC 24744:2007. International Organisation for Standardisation (ISO), Geneva, Switzerland, 2007.
- [5] *Software and Systems Process Engineering Meta-Model Specification, V2.0*. Number formal/2008-04-01. Object Management Group (OMG), MA, USA, April 2008.
- [6] *ARP4754A, Guidelines for Development of Civil Aircraft and Systems*. SAE International, Warrendale, PA, USA, 2010.
- [7] *Business Process Model and Notation, version 2.0*. Number formal/2011-01-03. Object Management Group (OMG), MA, USA, January 2011.
- [8] *GSN: Community Standard Version 1*. Origin Consulting (York) Limited, United Kingdom, 2011.
- [9] *SACM: Structured Assurance Case Metamodel, Version 1.0*. Number formal/2013-02-01. Object Management Group (OMG), MA, USA, February 2013.
- [10] *Kernel And Language For Software Engineering Methods (Essence), V1.1*. Number formal/2015-12-02. Object Management Group (OMG), MA, USA, December 2015.
- [11] S Abhishek and M Frank. A Roadmap for Software Engineering for the Cloud: Results of a Systematic Review. In X Wang, N Ali, I Ramos, and R Vidgen, editors, *Agile and Lean Service-Oriented Development: Foundations, Theory, and Practice*, pages 48–63. IGI Global, 2012.
- [12] S Acuña, A De Antonio, X Ferré, M López, and L Maté. The software process: Modelling, evaluation and improvement. In S. K Chang, editor, *Handbook of Software Engineering and Knowledge Engineering, World Scientific*, volume 1, pages 193–237. World Scientific, 2001.
- [13] S Alajrami, B Gallina, and A Romanovsky. EXE-SPEM: Towards Cloud-based Executable Software Process Models. In *MODELSWARD'16 - Proceedings of the 4rd International Conference on Model-Driven Engineering and Software Development, Rome, Italy, 19-21 February.*, pages 517–526. Scitepress, 2016.

- [14] S Alajrami, B Gallina, I Sljivo, A Romanovsky, and P Isberg. Towards Cloud-Based Enactment of Safety-Related Processes. In A Skavhaug, J Guiochet, and F Bitsch, editors, *Computer Safety, Reliability, and Security - 35th International Conference, SAFECOMP'16, Trondheim, Norway, September 21-23, Proceedings*, pages 309–321. Springer, 2016.
- [15] S Alajrami, A Romanovsky, and B Gallina. Software Development in the Post-PC Era: Towards Software Development as a Service. In P Abrahamsson and A Jedlitschka, editors, *The 17th International Conference on Product-Focused Software Process Improvement, PROFES'16, Trondheim, Norway, November 22-24, Proceedings*. Springer, 2016.
- [16] A Arunthavanathan, S Shanmugathan, S Ratnavel, V Thiyagarajah, I Perera, D Meedeniya, and D Balasubramaniam. Support for traceability management of software artefacts using Natural Language Processing. In *Moratuwa Engineering Research Conference (MERCon), Katubedda, Sri Lanka April 5-6, Proceedings*, pages 18–23. IEEE, April 2016.
- [17] M Azoff. White paper: The Benefits of Model Driven Development. MDD in Modern Web Based Systems. Technical report, Butler Direct Limited, Hull, East Yorkshire, UK, March 2008.
- [18] R Bahsoon, N Ali, I Mistrík, and T. S Mohan. The iee services track on software engineering for/in the cloud. In R Bahsoon and L.-J Zhang, editors, *Proceedings of the 2016 IEEE World Congress on Services (SERVICES), San Francisco, USA, June 27 - July 2*, pages 97–98, June 2016.
- [19] R Bahsoon, I Mistrík, N Ali, T Mohan, and N Medvidović. The future of software engineering in and for the cloud. *Journal of Systems and Software*, 86(9):2221–2224, September 2013.
- [20] A Bala and I Chana. Article: A Survey of Various Workflow Scheduling Algorithms in Cloud Environment. *IJCA Proceedings on 2nd National Conference on Information and Communication Technology*, NCICT(4):26–30, November 2011.
- [21] T Ball, S Burckhardt, J de Halleux, M Moskal, and N Tillmann. Beyond Open Source: The TouchDevelop Cloud-based Integrated Development Environment. Technical Report MSR-TR-2014-127, Microsoft Research, September 2014.
- [22] S. C Bandinelli, A Fuggetta, and C Ghezzi. Software process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering*, 19(12):1128–1144, 1993.
- [23] J Barnat, L Brim, V Havel, J Havlíček, J Kriho, M Lenčo, P Ročkai, V Štill, and J Weiser. DiVinE 3.0 – An Explicit-State Model Checker for Multithreaded C & C++ Programs. In N Sharygina and H Veith, editors, *Computer Aided Verification - 25th International Conference, CAV'13, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *LNCS*, pages 863–868. Springer, 2013.
- [24] A Barnes and J Gray. COTS, workflow, and software process management: an exploration of software engineering tool development. In *12th Australian Software Engineering Conference (ASWEC 2000), April 28-30, Canberra, Australia*, pages 221–232. IEEE Computer Society, 2000.

- [25] R Bendraou, J Jezequel, M.-P Gervais, and X Blanc. A Comparison of Six UML-Based Languages for Software Process Modelling. *IEEE Transactions on Software Engineering*, 36(5):662–675, Sept 2010.
- [26] M Biehl and W Löwe. Automated Architecture Consistency Checking for Model Driven Software Development. In R Mirandola, I Gorton, and C Hofmeister, editors, *Architectures for Adaptive Software Systems, 5th International Conference on the Quality of Software Architectures, QoSA 2009, East Stroudsburg, PA, USA, June 24-26, 2009, Proceedings*, pages 36–51. Springer, 2009.
- [27] P Bishop and R Bloomfield. A Methodology for Safety Case Development. In F Redmill and T Anderson, editors, *Industrial Perspectives of Safety-critical Systems: 6th Safety-critical Systems Symposium, Birmingham, UK*, pages 194–203. Springer, 1998.
- [28] B Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, May 1988.
- [29] B Boehm. Some future trends and implications for systems and software engineering processes. *Systems Engineering*, 9(1):1–19, 2006.
- [30] F Brooks. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4):10–19, April 1987.
- [31] S Bucur, V Ureche, C Zamfir, and G Candea. Parallel Symbolic Execution for Automated Real-world Software Testing. In *Proceedings of the Sixth European conference on Computer systems, EuroSys'11, Salzburg, Austria, April 10-13*, pages 183–198. ACM, 2011.
- [32] E Carmel. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.
- [33] D Chan and K Leung. Software development as a workflow process. In *4th Asia-Pacific Software Engineering and International Computer Science Conference (APSEC '97 / ICSC '97), December 2-5, Clear Water Bay, Hong Kong*, pages 282–291. IEEE Computer Society, 1997.
- [34] M. A Chauhan and M. A Babar. Cloud Infrastructure for Providing Tools As a Service: Quality Attributes and Potential Solutions. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA/ECSA'12, Helsinki, Finland, August 20-24 - Companion Volume*, pages 5–13, 2012.
- [35] B Combemale, X Crégut, A Caplain, and B Coulette. Towards a Rigorous Process Modelling with SPEM. In Y Manolopoulos, J Filipe, P Constantopoulos, and J Cordeiro, editors, *ICEIS - Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration, Paphos, Cyprus, May 23-27*, pages 530–533. INSTICC Press, 2006.
- [36] E. O Conchúir, P Ågerfalk, H Olsson, and B Fitzgerald. Global Software Development: Where Are the Benefits? *Commun. ACM*, 52(8):127–131, August 2009.

- [37] R Conradi, M. L Jaccheri, C Mazzi, M. N Nguyen, and A Aarsten. Design, Use and Implementation of SPELL, a Language for Software Process Modelling and Evolution. In *Software Process Technology, Second European Workshop, EWSPT '92, Trondheim, Norway, September 7-8, Proceedings*, pages 167–177, 1992.
- [38] G. C. B Costa, C. M. L Werner, and R Braga. Software Process Performance Improvement Using Data Provenance and Ontology. In M La Rosa, P Loos, and O Pastor, editors, *Business Process Management Forum - BPM Forum 2016, Rio de Janeiro, Brazil, September 18-22, Proceedings*, pages 55–71. Springer, 2016.
- [39] B Curtis, M. I Kellner, and J Over. Process Modeling. *Commun. ACM*, 35(9):75–90, September 1992.
- [40] R Dawson, P Bones, B. J Oates, P Brereton, M Azuma, and M. L Jackson. Empirical Methodologies in Software Engineering. In *11th International Workshop on Software Technology and Engineering Practice (STEP'03), 19-21 September 2003, Amsterdam, The Netherlands*, pages 52–58. IEEE Computer Society, 2003.
- [41] E. W Dijkstra. The Humble Programmer. *Commun. ACM*, 15(10):859–866, October 1972.
- [42] J.-L Doumont. Verbal versus visual: A word is worth a thousand pictures, too. *Technical communication*, 49(2):219–224, 2002.
- [43] P Dourish and V Bellotti. Awareness and Coordination in Shared Workspaces. In *CSCW '92, Proceedings of the Conference on Computer Supported Cooperative Work, Toronto, Canada, October 31 - November 4*, pages 107–114. ACM, 1992.
- [44] J. J Durillo and R Prodan. Multi-objective workflow scheduling in Amazon EC2. *Cluster Computing*, 17(2):169–189, 2014.
- [45] R Ellner, S Al-Hilank, J Drexler, M Jung, D Kips, and M Philippsen. eSPEM - A SPEM Extension for Enactable Behavior Modeling. In T Kühne, B Selic, M.-P Gervais, and F Terrier, editors, *Modelling Foundations and Applications, 6th European Conference, ECMFA 2010, Paris, France, June 15-18. Proceedings*, volume 6138 of *Lecture Notes in Computer Science*, pages 116–131. 2010.
- [46] R Ellner, S Al-Hilank, J Drexler, M Jung, D Kips, and M Philippsen. A FUML-Based Distributed Execution Machine for Enacting Software Process Models. In R. B France, J. M Kuester, B Bordbar, and R Paige, editors, *Modelling Foundations and Applications - 7th European Conference, ECMFA 2011, Birmingham, UK, June 6 - 9, 2011 Proceedings*, volume 6698 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.
- [47] B Elvesæter, G Benguria, and S Ilieva. A Comparison of the Essence 1.0 and SPEM 2.0 Specifications for Software Engineering Methods. In R Lbath, Bendraou, B Coulette, and M.-P Gervais, editors, *Proceedings of the Third Workshop on Process-Based Approaches for Model-Driven Engineering, PMDE@ECOOP'13, Montpellier, France, July 1*, PMDE '13, pages 2:1–2:10. ACM, 2013.
- [48] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of

personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119/59, May 2016.

- [49] D Feitelson, E Frachtenberg, and K Beck. Development and Deployment at Facebook. *IEEE Internet Computing*, 17(4):8–17, July 2013.
- [50] J Fiaidhi, I Bojanova, J Zhang, and L.-J Zhang. Enforcing Multitenancy for Cloud Computing Environments. *IT Professional*, 14(1):16–18, January 2012.
- [51] D. G Firesmith and B Henderson-Sellers. *The OPEN Process Framework: An Introduction*. OPEN series. Addison-Wesley, 2002.
- [52] A Fuggetta. Software Process: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering Co-located with the International Conference on Software Engineering ICSE'00, Limerick, Ireland, June 4-11*, pages 25–34. ACM, 2000.
- [53] A Fuggetta and E Di Nitto. Software Process. In *Proceedings of the on Future of Software Engineering, FOSE'14, Hyderabad, India, May 31 - June 7*, pages 1–12. ACM, 2014.
- [54] B Gallina, M. A Javed, F. U Muram, and S Punnekkat. Model-driven Dependability Analysis Method for Component-based Architectures. In *38th Euromicro Conference on Software Engineering and Advanced Applications, SEAA'12, Cesme, Izmir, Turkey, September 5-8*, pages 233–240. IEEE, 2012.
- [55] B Gallina, E Sefer, and A Refsdal. Towards safety risk assessment of socio-technical systems via failure logic analysis. In *2nd IEEE International Symposium on Software Reliability Engineering Workshops, joint event of ISSRE'14, Naples, Italy, November 3-6*, pages 287–292, 2014.
- [56] B Gallina. A Model-driven Safety Certification Method for Process Compliance. In *25th IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Naples, Italy, November 3-6*, pages 204–209. IEEE, 2014.
- [57] B Gallina, K Pitchai, and K Lundqvist. S-TunExSPEM: Towards an Extension of SPEM 2.0 to Model and Exchange Tunable Safety-Oriented Processes. In R Lee, editor, *Software Engineering Research, Management and Applications [selected papers from the 11th International Conference on Software Engineering Research, Management and Applications, SERA 2013, Prague, Czech Republic, August 7-9]*, volume 496 of *Studies in Computational Intelligence*, pages 215–230. Springer, 2014.
- [58] J. H Gerard. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.
- [59] J. H Gerard. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
- [60] C Gutwin, K Schneider, D Paquette, and R Penner. Supporting Group Awareness in Distributed Software Development. In R Bastide, P Palanque, and J Roth, editors, *Engineering Human Computer Interaction and Interactive Systems, Joint Working Conferences EHCI-DSVIS'04, Hamburg, Germany, July 11-13, Revised Selected Papers*, pages 383–397. Springer, 2004.

- [61] I Habli and T Kelly. Process and Product Certification Arguments: Getting the Balance Right. *SIGBED Rev.*, 3(4):1–8, 2006.
- [62] S. I Hashmi, V Clerc, M Razavian, C Manteli, D. A Tamburri, P Lago, E. D Nitto, and I Richardson. Using the cloud to facilitate global software development challenges. In *2011 IEEE Sixth International Conference on Global Software Engineering Workshop*, pages 70–77, Aug 2011.
- [63] C. L Heitmeyer, R. D Jeffords, and B. G Labaw. Automated Consistency Checking of Requirements Specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, July 1996.
- [64] B Henderson-Sellers and C Gonzalez-Perez. A comparison of four process meta-models and the creation of a new generic standard. *Information and Software Technology*, 47(1):49 – 65, 2005.
- [65] J. D Herbsleb. Global Software Engineering: The Future of Socio-technical Coordination. In L. C Briand and A. L Wolf, editors, *International Conference on Software Engineering, ISCE’07, Workshop on the Future of Software Engineering, FOSE’07, May 23-25, Minneapolis, MN, USA*, pages 188–198. IEEE Computer Society, 2007.
- [66] H Hiden, S Woodman, P Watson, and J Cala. Developing cloud applications using the e-Science Central platform. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 371(1983), 2012.
- [67] D Hollingsworth. *Workflow Reference Model*. Number TC00-1003. Workflow Management Coalition (WfMC), January 1995.
- [68] C. M Holloway. Safety Case Notations: Alternatives for the Non-Graphically Inclined? In *3rd IET International Conference on System Safety, October 20-22, Birmingham, UK*, pages 1–6. IET, 2008.
- [69] C. M Holloway. Explicate ’78: Uncovering the Implicit Assurance Case in DO-178C. Technical Report 20150009473, NASA Langley Research Centre, USA, 2015.
- [70] J Humble and D Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 1st edition, 2010.
- [71] S Jang, X Wu, V Taylor, G Mehta, K Vahi, and E Deelman. Using performance prediction to allocate grid resources. Technical Report GriPhyN Project, TR 2004-25, Texas A&M University, USA, 2004.
- [72] R. W Jensen. *Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management*. Pearson Education, 2014.
- [73] H Kagdi, M. L Collard, and J. I Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):77–131, 2007.
- [74] G. E Kaiser, N. S Barghouti, and M. H Sokolsky. Preliminary experience with process modelling in the MARVEL software development environment kernel. In *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences, Kailua-Kona, HI, USA. January 2-5*, volume 2, pages 131–140, 1990.

- [75] R Kazman, L Bass, M Webb, and G Abowd. SAAM: A Method for Analyzing the Properties of Software Architectures. In *Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 16-21, ICSE '94*, pages 81–90. IEEE Computer Society Press, USA, 1994.
- [76] R Kazman, M Klein, and P Clements. ATAM: Method for Architecture Evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, USA, 2000.
- [77] M Kellner, R Madachy, and D Raffo. Software Process Simulation Modeling: Why? What? How? *Journal of Systems and Software*, 46(2-3):91 – 105, 1999.
- [78] G Kim, K Behr, and G Spafford. *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*. IT Revolution Press, 1st edition, 2013.
- [79] M Kuhrmann, D. M Fernández, and R Steenweg. Systematic Software Process Development: Where Do We Stand Today? In D Notkin, B. H. C Cheng, and K Pohl, editors, *International Conference on Software and System Process, ICSSP '13, San Francisco, CA, USA, May 18-19*, pages 166–170. ACM, 2013.
- [80] F Lanubile, F Calefato, and C Ebert. Group Awareness in Global Software Engineering. *IEEE Software*, 30(2):18–23, March 2013.
- [81] N Lassing, D Rijsenbrij, and H van Vliet. On Software Architecture Analysis of Flexibility, Complexity of Changes: Size Isn't Everything. In *Proceedings of the Second Nordic Software Architecture Workshop (NOSA '99), Ronneby, Sweden, August 12-13*. Department of Software Engineering and Computer Science/Blekinge Institute of Technology, Sweden, 1999.
- [82] K Liu, H Jin, J Chen, X Liu, D Yuan, and Y Yang. A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on Cloud Computing Platform. *International Journal of High Performance Computing Applications*, 24:445–456, November 2010.
- [83] L Maciaszek and B Liang. *Practical Software Engineering: A Case-Study Approach*. Addison-Wesley, 2004.
- [84] M Mao and M Humphrey. Scaling and Scheduling to Maximise Application Performance within Budget Constraints in Cloud Workflows. In *IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS), May 20-24, Boston, Massachusetts USA*, pages 67–78. CPS, May 2013.
- [85] M Mao and M Humphrey. Auto-scaling to Minimise Cost and Meet Application Deadlines in Cloud Workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, November 12-18, SC'11, Seattle, WA, USA*, pages 49:1–49:12. ACM, 2011.
- [86] E. M Maximilien and P Campos. Facts, Trends and Challenges in Modern Software Development. *International Journal of Agile and Extreme Software Development*, 1(1):1–5, July 2012.

- [87] S Meilin, Y Guangxin, X Yong, and W Shangguang. Workflow management systems: a survey. In X Chunpei, editor, *International Conference on Communication Technology (ICCT '98) Proceedings, Beijing, China, October 22-24*, volume 2, pages S33-05-1 – S33-05-6. Publishing House of Construction Materials, IEEE, October 1998.
- [88] J Münch, O Armbrust, M Kowalczyk, and M Soto. *Software Process Definition and Management*. Springer, 2012.
- [89] P Naur and B Randell, editors. *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Belgium, Scientific Affairs Division, NATO*. 1969.
- [90] A Oberweis. Workflow Management In Software Engineering Projects. In S Medhat, editor, *Proceedings of the 2nd International Conference on Concurrent Engineering and Electronic Design Automation, April 7-8, Bournemouth, United Kingdom*, pages 55-60. Society for Computer Simulation, 1994.
- [91] M Oriol and F Ullah. YETI on the Cloud. In *Third International Conference on Software Testing, Verification and Validation, ICST'10, Paris, France, April 7-9, Workshops Proceedings*, pages 434-437, 2010.
- [92] M Paulk, W Curtis, M. B Chrissis, and C Weber. Capability Maturity Model for Software (Version 1.1). Technical Report CMU/SEI-93-TR-024, Software Engineering Institute, Carnegie Mellon University, USA, 1993.
- [93] C Portela, A Vasconcelos, A Silva, E Silva, M Gomes, M Ronny, W Lira, and S Oliveira. xSPIDER_ML: Proposal of a Software Processes Enactment Language Compliant with SPEM 2.0. *Journal of Software Engineering and Applications*, 5(6):375 – 384, 2012.
- [94] M Rahman, R Hassan, R Ranjan, and R Buyya. Adaptive workflow scheduling for dynamic grid and cloud computing environment. *Concurrency and Computation: Practice and Experience*, 25(13):1816-1842, 2013.
- [95] W Royce. Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings of the 9th International Conference on Software Engineering (ICSE'87), California, USA, March 30 - April 2*, pages 328-338. IEEE Computer Society Press, USA, 1987.
- [96] W Royce. Current Problems. In C Anderson and M Dorfman, editors, *Aerospace Software Engineering: A Collection of Concepts*. American Institute of Aeronautics, Inc., Washington DC, 1991.
- [97] P Runeson, M Host, A Rainer, and B Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley Publishing, 1st edition, 2012.
- [98] J Rushby. New Challenges in Certification for Aircraft Software. In *Proceedings of the 11th International Conference on Embedded Software, EMSOFT'11, part of the Seventh Embedded Systems Week, ESWeek'11, Taipei, Taiwan, October 9-14*, EMSOFT, pages 211-218, 2011.

- [99] D. C Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.
- [100] E Silva and D Lucrédio. Software Engineering for the Cloud: A Research Roadmap. In *26th Brazilian Symposium on Software Engineering, SBES'12, Natal, Brazil, September 23-28*, pages 71–80, Sept 2012.
- [101] L Singh and S Singh. Article: A Survey of Workflow Scheduling Algorithms and Research Issues. *International Journal of Computer Applications*, 74(15):21–28, July 2013.
- [102] I Sljivo, B Gallina, J Carlson, H Hansson, and S Puri. A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis. In *Software Reuse for Dynamic Systems in the Cloud and Beyond - 14th International Conference on Software Reuse, ICSR'15, Miami, FL, USA, January 4-6, Proceedings*, pages 253–268. Springer, 2015.
- [103] W Smith, I. T Foster, and V. E Taylor. Predicting Application Run Times Using Historical Information. In *Job Scheduling Strategies for Parallel Processing, IPP-S/SPDP'98 Workshop, Orlando, Florida, USA, March 30, Proceedings*, pages 122–142, London, UK, 1998. Springer.
- [104] R Steenweg, M Kuhrmann, and D. M Fernández. Software Engineering Process Metamodels. Technical Report TUM-I1220, Technical University of Munich, Germany, 2012.
- [105] W van der Aalst and A ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.
- [106] M. Y Vardi and P Wolper. Reasoning about Infinite Computations. *Information and Computation*, 115(1):1 – 37, 1994.
- [107] W. E Vesel, F. F Goldberg, N. H Roberts, and D. F Haasl. *Fault Tree Handbook (NUREG-0492)*. United States Nuclear Regulatory Commission, USA, January 1981.
- [108] Vijindra and S Shenai. Survey on Scheduling Issues in Cloud Computing. *International Conference on Modelling Optimization and Computing, Procedia Engineering*, 38:2881 – 2888, 2012.
- [109] J Wang, P Korambath, I Altintas, J Davis, and D Crawl. Workflow as a Service in the Cloud: Architecture and Scheduling Algorithms. *Procedia Computer Science*, 29:546 – 556, 2014.
- [110] WFMC. *Workflow Management Coalition Terminology & Glossary , Issue 3*. Number WFMC-TC-1011. February 1999.
- [111] WFMC. *XML Process Definition Language 2.2*. Number WFMC-TC-1025. August 2012.
- [112] K Wolstencroft, R Haines, D Fellows, A Williams, D Withers, S Owen, S Soiland-Reyes, I Dunlop, A Nenadic, P Fisher, J Bhagat, K Belhajjame, F Bacall, A Hardisty, A Nieva de la Hidalgo, M. P Balcazar Vargas, S Sufi, and C Goble. The Taverna

- workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 2013.
- [113] Z Wu, X Liu, Z Ni, D Yuan, and Y Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256–293, 2013.
- [114] Z Xiao and Y Xiao. Security and Privacy in Cloud Computing. *IEEE Communications Surveys Tutorials*, 15(2):843–859, February 2013.
- [115] J Yu, R Buyya, and C. K Tham. Cost-based scheduling of scientific workflow applications on utility grids. In H Stockinger, R Buyya, and R Perrott, editors, *First International Conference on e-Science and Grid Computing (e-Science’05), December 5-8, Melbourne, Australia*, pages 140–147, July 2005.
- [116] J Yu and R Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. *SIGMOD Rec.*, 34(3):44–49, September 2005.
- [117] F Yuan, M Li, and Z Wan. SPEM2XPDL: Towards SPEM Model Enactment. In H. R Arabnia, H Reza, L Deligiannidis, J. J Cuadrado-Gallego, V Schmidt, and A. M. G Solo, editors, *Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP’06, Las Vegas, Nevada, USA, June 26-29, Volume 1*, pages 240–245, 2006.

Appendices

B

THE XML PROCESS MODEL FOR FACEBOOK'S CONTINUOUS DELIVERY PROCESS

```
<?xml version="1.0" encoding="UTF-8"?>
<Process xmlns="http://mycompany.com/namespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" ID="ID1" xsi:schemaLocation="http://mycompany.com/namespace_Process-v1.2.xsd">
  <Description>Facebook Process</Description>
  <Start_with>560c154ae4b0564154k3e5bb</Start_with>
  <Elements>
    <Activity ID="560c154ae4b0564154k3e5bb">
      <Name>Code Dev</Name>
      <Description>editing source code files</Description>
      <Interactive>true</Interactive>
      <No_of_input_ports>0</No_of_input_ports>
      <No_of_output_ports>1</No_of_output_ports>
      <Wait_for_input>false</Wait_for_input>
      <Responsible_role>Engineer 1</Responsible_role>
      <Version>1</Version>

      <Out_ports>
        <Out_port>
          <Next_activity>560c154ae4b0564154k3z4fs</Next_activity>
          <Artefact ID="560c154ae4b0564156c5d5ba">
            <Filename>source_code</Filename>
            <Filetype>zip</Filetype>
            <Description>the source code for the new feature
            </Description>
            <Version>1</Version>
          </Artefact>
        </Out_port>
      </Out_ports>
    </Activity>

    <Activity ID="560c154ae4b0564154k3z4fs">
      <Name>Phabricator Review</Name>
      <Description>checking source code files</Description>
      <Interactive>true</Interactive>
      <No_of_input_ports>1</No_of_input_ports>
      <No_of_output_ports>2</No_of_output_ports>
      <Wait_for_input>true</Wait_for_input>
      <Responsible_role>Engineer 2</Responsible_role>
      <Version>1</Version>

      <In_ports>
        <In_port>
          <From_activity>560c154ae4b0564154k3e5bb</From_activity>
          <Artefact ID="560c154ae4b0564156c5d5ba">
            <Filename>source_code</Filename>
            <Filetype>zip</Filetype>
            <Description>the source code for the new feature</Description>
            <Version>1</Version>
          </Artefact>
        </In_port>
      </In_ports>
      <Out_ports>
        <Out_port>
          <Next_activity>560c154ae4b0564154k3e5bb</Next_activity>
```


Appendix B: The XML Process Model for Facebook's Continuous Delivery Process

```

        <Artefact ID="560c154ae4b0564156c5h8nx">
            <Filename>fix_requests</Filename>
            <Filetype>doc</Filetype>
            <Description>requests for bug fixing</Description>
            <Version>1</Version>
        </Artefact>
    </Out_port>
    <Out_port>
        <Next_activity>560c154ae4b0564154k3kl2d</Next_activity>
        <Artefact ID="560c154ae4b0564156c5d5ba">
            <Filename>source_code</Filename>
            <Filetype>zip</Filetype>
            <Description>the source code for the new feature</Description>
            <Version>2</Version>
        </Artefact>
    </Out_port>
</Out_ports>
</Activity>

<Activity ID="560c154ae4b0564154k3kl2d">
    <Name>Regression Testing</Name>
    <Description>integrating the code and performing regression testing</
Description>
    <Interactive>>false</Interactive>
    <No_of_input_ports>1</No_of_input_ports>
    <No_of_output_ports>2</No_of_output_ports>
    <Wait_for_input>>true</Wait_for_input>
    <Responsible_role>Engineer 2</Responsible_role>
    <Version>1</Version>

    <In_ports>
        <In_port>
            <From_activity>560c154ae4b0564154k3z4fs</From_activity>
            <Artefact ID="560c154ae4b0564156c5d5ba">
                <Filename>source_code</Filename>
                <Filetype>zip</Filetype>
                <Description>the source code for the new feature</Description>
                <Version>2</Version>
            </Artefact>
        </In_port>
    </In_ports>
    <Out_ports>
        <Out_port>
            <Next_activity>560c154ae4b0564154k3kl2d</Next_activity>
            <Artefact ID="560c154ae4b0564156c5si7q">
                <Filename>bug_fixes</Filename>
                <Filetype>doc</Filetype>
                <Description>requests for bug fixing</Description>
                <Version>1</Version>
            </Artefact>
        </Out_port>
        <Out_port>
            <Next_activity>560c154ae4b0564154k3jl9e;560c154ae4b0564154k3ld4r
            </Next_activity>
            <Artefact ID="560c154ae4b0564156c5d5ba">
                <Filename>source_code</Filename>
                <Filetype>zip</Filetype>
                <Description>the source code for the new feature</Description>
                <Version>3</Version>
            </Artefact>
        </Out_port>
    </Out_ports>
    <Cloud_config>
        <Cloud_deployment_model>public</Cloud_deployment_model>
        <Cloud_provider>AWS</Cloud_provider>
        <Instance_type>m3.xlarge</Instance_type>
        <No_of_instances>2</No_of_instances>
        <Timeout>2</Timeout>
    </Cloud_config>
</Activity>

<Activity ID="560c154ae4b0564154k3jl9e">
    <Name>Internal Release</Name>
    <Description>releasing the new feature for internal use</Description>

```

Appendix B: The XML Process Model for Facebook's Continuous Delivery Process

```
<Interactive>>false</Interactive>
<No_of_input_ports>1</No_of_input_ports>
<No_of_output_ports>1</No_of_output_ports>
<Wait_for_input>>true</Wait_for_input>
<Responsible_role>Engineer 3</Responsible_role>
<Version>1</Version>

<In_ports>
  <In_port>
    <From_activity>560c154ae4b0564154k3k12d</From_activity>
    <Artefact ID="560c154ae4b0564156c5d5ba">
      <Filename>source_code</Filename>
      <Filetype>zip</Filetype>
      <Description>the source code for the new feature</Description>
      <Version>3</Version>
    </Artefact>
  </In_port>
</In_ports>
<Out_ports>
  <Out_port>
    <Next_activity>560c154ae4b0564154k3ld4r</Next_activity>
    <Artefact ID="560c154ae4b0564156c5d5ba">
      <Filename>source_code</Filename>
      <Filetype>zip</Filetype>
      <Description>the source code for the new feature</Description>
      <Version>3</Version>
    </Artefact>
  </Out_port>
</Out_ports>
</Activity>

<Activity ID="560c154ae4b0564154k3ld4r">
  <Name>Preflab Testing</Name>
  <Description>performing tests using Preflab</Description>
  <Interactive>>false</Interactive>
  <No_of_input_ports>1</No_of_input_ports>
  <No_of_output_ports>1</No_of_output_ports>
  <Wait_for_input>>true</Wait_for_input>
  <Responsible_role>Engineer 3</Responsible_role>
  <Version>1</Version>

  <In_ports>
    <In_port>
      <From_activity>560c154ae4b0564154k3k12d</From_activity>
      <Artefact ID="560c154ae4b0564156c5d5ba">
        <Filename>source_code</Filename>
        <Filetype>zip</Filetype>
        <Description>the source code for the new feature</Description>
        <Version>3</Version>
      </Artefact>
    </In_port>
  </In_ports>
  <Out_ports>
    <Out_port>
      <Next_activity>560c154ae4b0564154k3ld4r</Next_activity>
      <Artefact ID="560c154ae4b0564156c5d5ba">
        <Filename>source_code</Filename>
        <Filetype>zip</Filetype>
        <Description>the source code for the new feature</Description>
        <Version>4</Version>
      </Artefact>
    </Out_port>
  </Out_ports>
</Activity>

<Activity ID="560c154ae4b0564154k3ld4r">
  <Name>Deploy 1</Name>
  <Description>deploy new feature to internal servers only</Description>
  <Interactive>>false</Interactive>
  <No_of_input_ports>2</No_of_input_ports>
  <No_of_output_ports>1</No_of_output_ports>
  <Wait_for_input>>true</Wait_for_input>
  <Responsible_role>Engineer 4</Responsible_role>
```

Appendix B: The XML Process Model for Facebook's Continuous Delivery Process

```
<Version>1</Version>

<In_ports>
  <In_port>
    <From_activity>560c154ae4b0564154k3jl9e</From_activity>
    <Artefact ID="560c154ae4b0564156c5d5ba">
      <Filename>source_code</Filename>
      <Filetype>zip</Filetype>
      <Description>the source code for the new feature</Description>
      <Version>3</Version>
    </Artefact>
  </In_port>
  <In_port>
    <From_activity>560c154ae4b0564154k3ld4r</From_activity>
    <Artefact ID="560c154ae4b0564156c5d5ba">
      <Filename>source_code</Filename>
      <Filetype>zip</Filetype>
      <Description>the source code for the new feature</Description>
      <Version>4</Version>
    </Artefact>
  </In_port>
</In_ports>
<Out_ports>
  <Out_port>
    <Next_activity>560c154ae4b0564154k3bv4t</Next_activity>
    <Artefact ID="560c154ae4b0564156c5d5ba">
      <Filename>source_code</Filename>
      <Filetype>zip</Filetype>
      <Description>the source code for the new feature</Description>
      <Version>5</Version>
    </Artefact>
  </Out_port>
</Out_ports>
</Activity>

<Activity ID="560c154ae4b0564154k3bv4t">
  <Name>Deploy 2</Name>
  <Description>deploy new feature to 1% of global servers</Description>
  <Interactive>>false</Interactive>
  <No_of_input_ports>1</No_of_input_ports>
  <No_of_output_ports>1</No_of_output_ports>
  <Wait_for_input>>true</Wait_for_input>
  <Responsible_role>Engineer 4</Responsible_role>
  <Version>1</Version>

  <In_ports>
    <In_port>
      <From_activity>560c154ae4b0564154k3ld4r</From_activity>
      <Artefact ID="560c154ae4b0564156c5d5ba">
        <Filename>source_code</Filename>
        <Filetype>zip</Filetype>
        <Description>the source code for the new feature</Description>
        <Version>5</Version>
      </Artefact>
    </In_port>
  </In_ports>
  <Out_ports>
    <Out_port>
      <Next_activity>560c154ae4b0564154k3sl8r</Next_activity>
      <Artefact ID="560c154ae4b0564156c5d5ba">
        <Filename>source_code</Filename>
        <Filetype>zip</Filetype>
        <Description>the source code for the new feature</Description>
        <Version>6</Version>
      </Artefact>
    </Out_port>
  </Out_ports>
</Activity>

<Activity ID="560c154ae4b0564154k3sl8r">
  <Name>Deploy 3</Name>
  <Description>deploy new feature to all servers</Description>
  <Interactive>>false</Interactive>
  <No_of_input_ports>1</No_of_input_ports>
```

Appendix B: The XML Process Model for Facebook's Continuous Delivery Process

```
<No_of_output_ports>1</No_of_output_ports>
<Wait_for_input>true</Wait_for_input>
<Responsible_role>Engineer 4</Responsible_role>
<Version>1</Version>

<In_ports>
  <In_port>
    <From_activity>560c154ae4b0564154k3bv4t</From_activity>
    <Artefact_ID="560c154ae4b0564156c5d5ba">
      <Filename>source_code</Filename>
      <Filetype>zip</Filetype>
      <Description>the source code for the new feature</Description>
      <Version>6</Version>
    </Artefact>
  </In_port>
</In_ports>
<Out_ports>
  <Out_port>
    <Next_activity></Next_activity>
    <Artefact_ID="560c154ae4b0564156c5d5ba">
      <Filename>source_code</Filename>
      <Filetype>zip</Filetype>
      <Description>the source code for the new feature</Description>
      <Version>7</Version>
    </Artefact>
  </Out_port>
</Out_ports>
</Activity>

<Control_Point_ID="560d33bbe4b02110e8jm2n16">
  <Message>Choose whether to switch the feature on to all or subset of users.
</Message>
  <Options>
    <Option>
      <ActivityID>560c154ae4b0564154k3xy1q</ActivityID>
      <Parameter>allOrSubset</Parameter>
    </Option>
  </Options>
</Control_Point>

<Activity_ID="560c154ae4b0564154k3xy1q">
  <Name>Gatekeeper</Name>
  <Description>switch the feature on to all or subset of users</Description>
  <Interactive>>false</Interactive>
  <No_of_input_ports>1</No_of_input_ports>
  <No_of_output_ports>0</No_of_output_ports>
  <Wait_for_input>true</Wait_for_input>
  <Responsible_role>Engineer 4</Responsible_role>
  <Version>1</Version>

  <In_ports>
    <In_port>
      <From_activity>560d33bbe4b02110e8jm2n16</From_activity>
      <Parameter>allOrSubset</Parameter>
    </In_port>
  </In_ports>
</Activity>

</Elements>
</Process>
```

C

ARP4761

Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment- ARP4761- is an Aerospace Recommended Practice from SAE International. It was proposed to simplify and clarify the Federal Aviation Administration (FAA) advisory circular AC 25.1309-1A [1]. It is intended to be used in conjunction with ARP4754A. Both standards follow a functional approach to safety. ARP4761 is a document that provides guidance to perform safety assessment. More specifically, defines a set of partially ordered activities that need to be performed in support of the airworthiness process to handle hazardous events (system and equipment failure or malfunction that may lead to hazards). This set of partially ordered activities is known as Airworthiness Safety Assessment Process. The process is iterative and it starts with a high level design which is used to derive the safety requirements of the system. During the design development, the design evolves and so do the safety requirements. The safety assessment process verifies that the design meets the safety requirements and complies with the regulations.

Figure C.1 provides an overview of the Airworthiness Safety Assessment Process for aircraft. Safety assessment is an iterative process. As development phases evolve, the safety assessment evolves iteratively. For example, as the aircraft requirements evolve from the concept development phase to the preliminary design phase, new hazards and functions might be introduced. Therefore, the safety assessment evolves with the development cycle.

We focus on three processes within the Airworthiness Safety Assessment Process. The Functional Hazard Assessment (FHA) which identifies failure conditions in the system followed by Preliminary System Safety Assessment (PSSA) which evaluates the system design/architecture. Finally, System Safety Assessment (SSA) assess if the system design meets the safety requirements. The Airworthiness Safety Assessment Process also

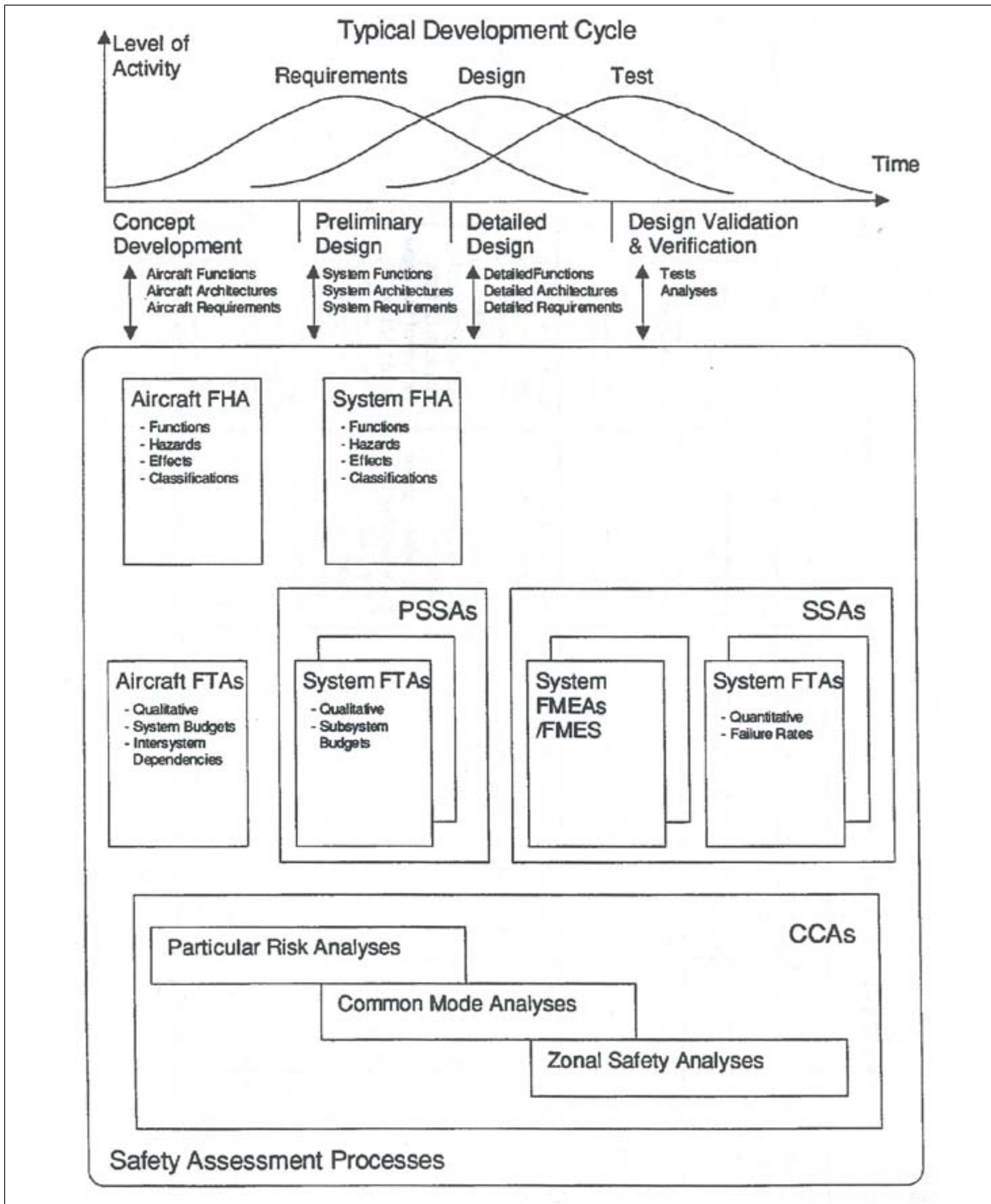


Figure C.1: An overview of the airworthiness safety assessment process [2]

includes the use of other techniques and tools for Common Cause Analysis (CCA) such as Zonal Safety Analysis (ZSA), Particular Risk Analysis (PRA) and Common Mode Analysis (CMA). Details about these techniques are beyond the scope of this thesis and are provided in the ARP4761 guidelines [2].

The FHA, PSSA and SSA processes are further detailed below:

- **Functional Hazard Assessment (FHA)**

FHA identifies the failure conditions in a system and classifies them based on their severity level. It is performed at both the aircraft and the system levels to examine the effect of individual and combined failures on the aircraft. FHA is a top-down process which starts at the aircraft level and proceeds to finer grained systems and subsystems. It starts with specifying high-level functional requirements (e.g., "*To control aircraft trajectory*") and identifying their associated failure conditions (e.g., "*Loss of aircraft control*"). This is then used to derive lower level requirements in an iterative manner. The identified failure conditions are then classified based on their severity ranging from a negligible/minor failure (a failure that does not have safety implications) to severe/catastrophic failures (failures which severely impacts the security of the aircraft). Based on this classification, tolerable limits of occurrence of these failures and Development Assurance Levels (DAL) are specified. The identified and classified failure conditions are passed as an input for the PSSA process.

- **Preliminary System Safety Assessment (PSSA)**

PSSA is conducted at multiple stages of the system development including system, item and hardware/software design definitions. It consists of a systematic examination of a proposed system architecture(s) to identify how system failures contribute to the failure conditions identified in the system FHA. It usually uses techniques like Fault Tree Analysis (FTA), Dependence Diagrams (DD) or Markov Analysis (MA) to identify system faults. PSSA takes in input the system FHA and the description of each system architecture under consideration. Based on the input received, the following set of tasks are performed within PSSA:

1. Completion of the list of aircraft and system level safety requirements,
2. Determination whether the received input architecture and planned concept design can reasonably be expected to meet the safety requirements and objectives,
3. Derivation of the safety requirements for the design of lower level items.

PSSA is focused on analysing the proposed system design and architecture to validate its safety. Moreover, PSSA includes identifying the derived safety re-

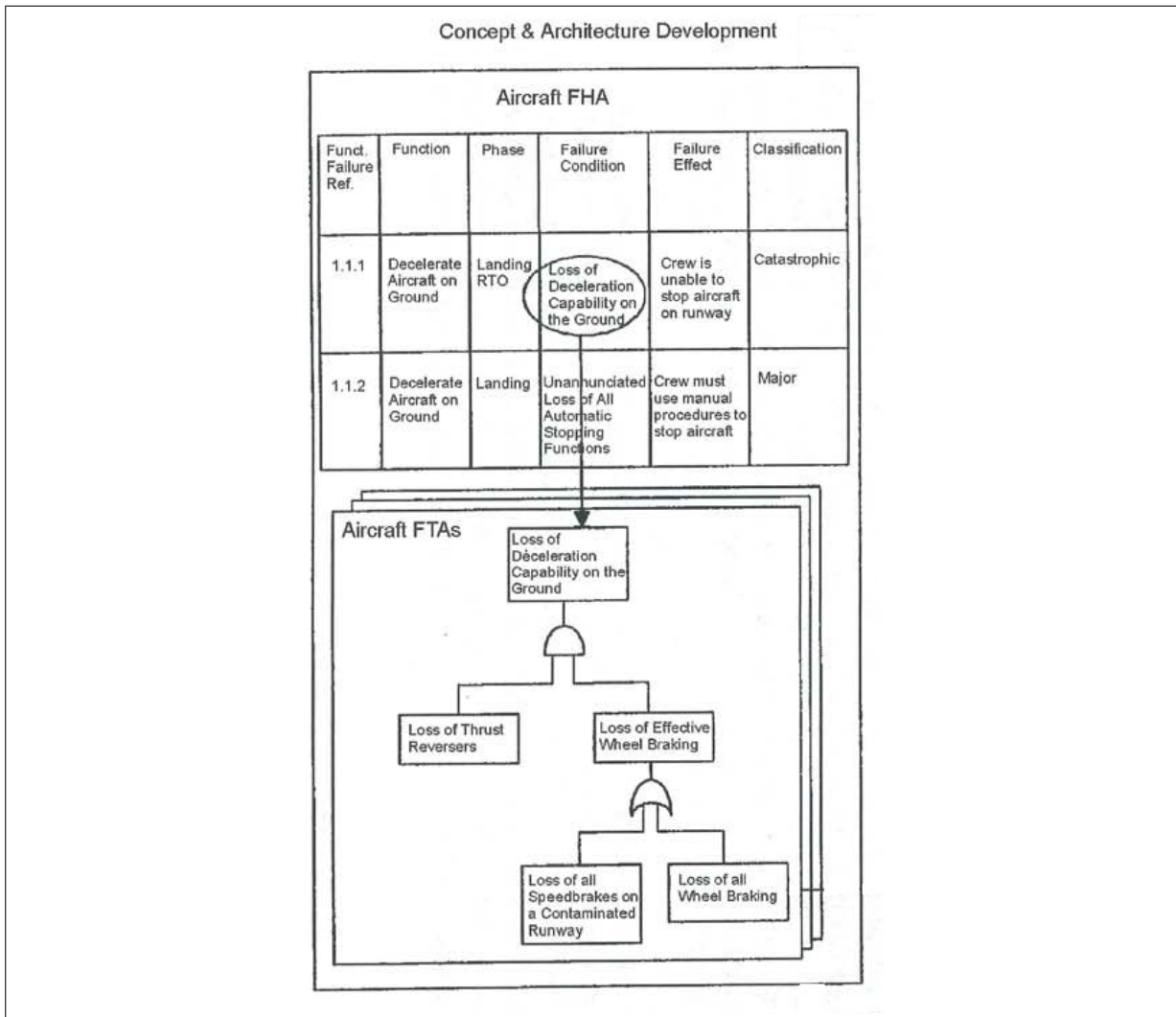


Figure C.2: Example of the relationship between FHA and FTA [2]

quirements, associating them with Development Assurance Levels (DALs) and allocating them to architectural elements. The final outcome of PSSA is: development failures effects of hardware and software, DALs, protective strategies and architectural features necessary to meet safety objectives. For further details, the reader may refer to Appendix B of ARP4761 [2].

Fault Tree Analysis (FTA) PSSA is usually conducted using FTA [107]. FTA is a deductive reasoning method for identifying root causes of hazardous (undesired) failures. A Fault Tree represents a group of events (parallel or sequential) and their interrelationships which can cause an undesired event of system failure. That undesired event is usually the root of the tree and is called *top event*. Events are categorised as: primary, intermediate and top events. Fault trees are usually graphically represented. Details about the different types of events and

the graphical symbols for the fault tree's building elements can be found in [107]. Figure C.2 illustrates an example of the aircraft FHA and its relationship with FTA. Each failure condition has an FTA which is then explored more in depth as the development of the system design proceeds.

- **System Safety Assessment (SSA)**

The SSA process evaluates if the final design and implementation of the system meet the safety requirements identified at FHA and PSSA.

D

ARP4761 WHEEL BRAKE SYSTEM

To demonstrate the Airworthiness Safety Assessment Process, ARP4761 includes a detailed example showing how the process can be applied to a small aircraft system. This system is the Aircraft Wheel Brake System (WBS) illustrated in Figure D.1. The system consists of mechanical components (e.g., valves and pedals) and a Brake System Control Unit (BSCU) which controls the operation of the brake through a hydraulic system connected to the wheels of the aircraft.

The BSCU (as illustrated in Figure D.2) contains sub-components and is connected to the input of the pedals and sends output signals to the hydraulic system which mechanically control the aircraft wheels. Internally, each BSCU sub-component takes input and produces output which contributes to the BSCU output. Failures may occur on the input ports and propagate through the internal sub-components. Failures may either be mitigated, propagated as they are or transformed to another failure condition by each sub-component.

In line with the PSSA, we focus is on the failure behaviour of the system to show that the unacceptable failures have been successfully mitigated. Fault Propagation and Transformation Calculus (FPTC) is a failure logic analysis allowing for the calculation of the system level failure behaviour based on the failure behaviour of the individual components. The propagation of failures from the inputs to the outputs of a component are captured via FPTC rules. For example, the FPTC rule *"I1.valueCoarse → O1.comission"* for a component with input I1 and output O1, states that when I1 port exhibits coarse (i.e., clearly detectable) value failure, then the output O1 port exhibits commission failure (i.e., O1 is provided when not supposed to). Such rules capture the system failure behaviour that should be considered in the corresponding product-based argument. The supported FPTC syntax is shown in Figure D.3.

Figure D.4 details the input/output ports of each component and show the possible

Appendix D: ARP4761 Wheel Brake System

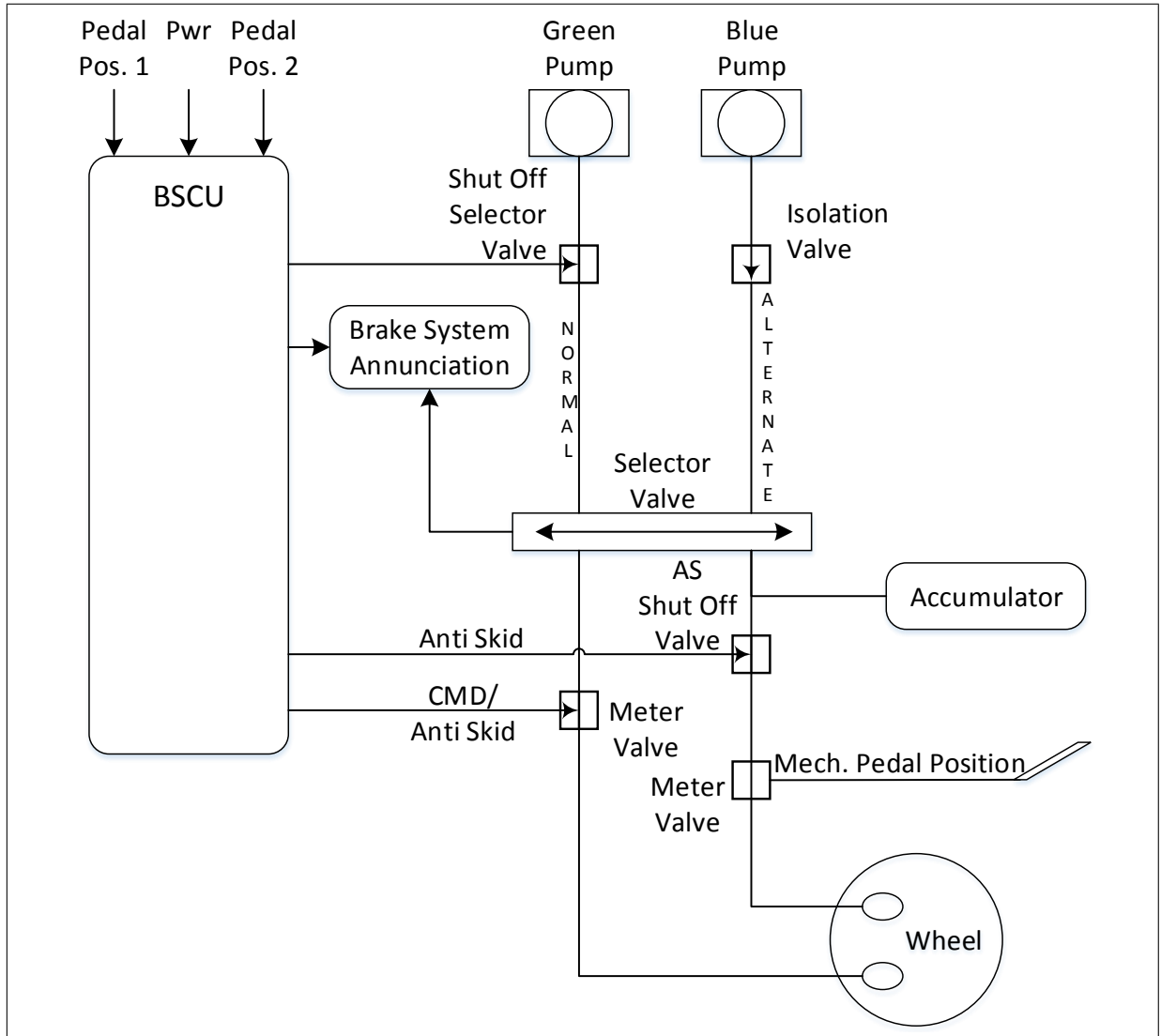


Figure D.1: The wheel brake system [2]

failures on the BSCU input and output ports.

The ARP4761 guidelines document demonstrates in detail how the Airworthiness Safety Assessment Process is applied to assess the safety of this system.

Appendix D: ARP4761 Wheel Brake System

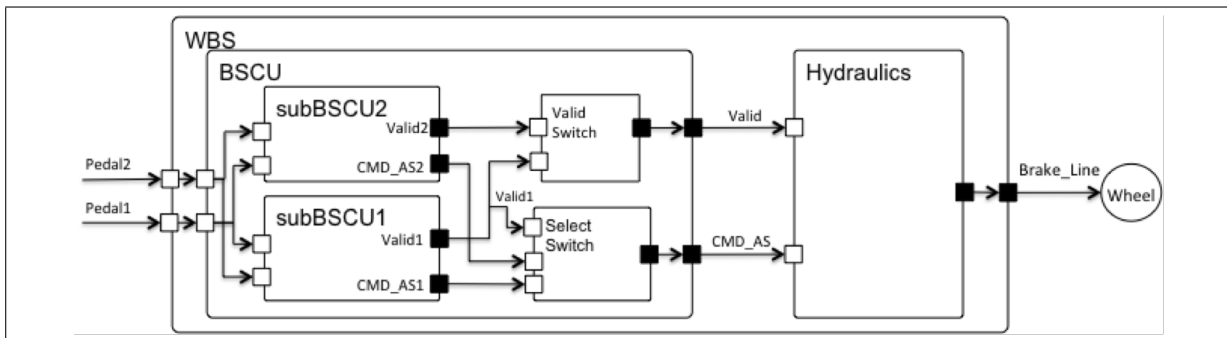


Figure D.2: The wheel brake system sub-components [2]

behaviour = expression + expression = LHS '→' RHS
LHS = portname '.' bL | portname '.' bL (' portname '.' bL) +
RHS = portname '.' bR | portname '.' bR (' portname '.' bR) +
failure = 'early' | 'late' | 'commission' | 'omission' | 'valueSubtle' | 'valueCoarse'
bL = 'wildcard' | bR
bR = 'noFailure' | failure

Figure D.3: FPTC syntax [54]

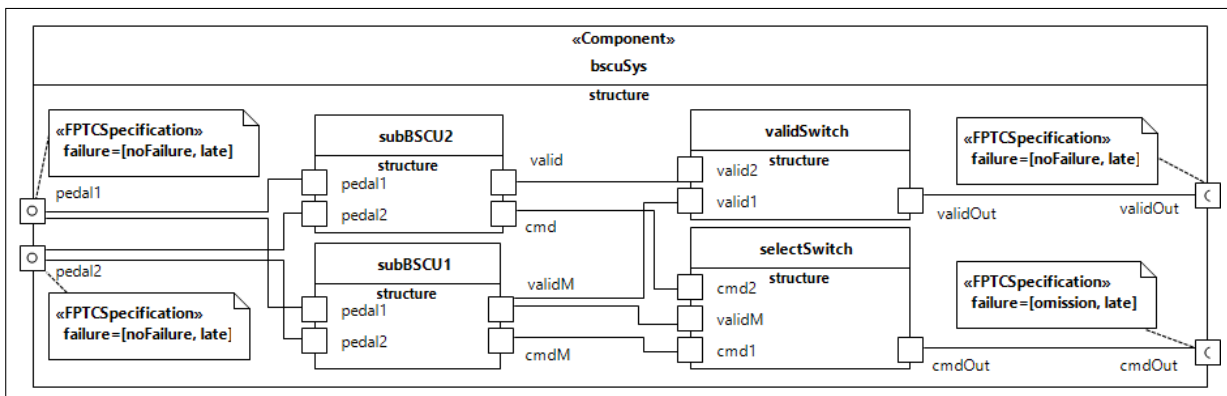


Figure D.4: The architectural model of BSCU components, ports and failures [2]

E

SAFETY CASE REPRESENTATION

Safety case argument fragments can be represented in different safety case representations, e.g., textually, graphically or machine-readable. These three types of representations are described below:

E.1 Visual representation

Goal Structuring Notation (GSN) [8] and Claims-Arguments-Evidence (CAE) [27] are the two main graphical notations for safety case representation. Here, we focus only on GSN.

Figure E.1 shows the basic GSN elements. The *goal* element is characterised by a statement representing a claim that should be supported by the underlying argument. *Strategies* can be used to describe the method used to develop a goal into additional sub-goals. The diamond symbol can be used to indicate that the goal needs further development. Different statements can be further clarified with the *context* element, while *solutions* are used to describe the evidence that the connected goal has been achieved. The *supportedBy* relationship is used to associate goals and strategies with other goals, strategies and solutions, while the *inContextOf* relationship is used to associate the goals, strategies and solutions with other supporting elements such as contexts.

Figure E.2 shows a simple GSN based argument fragment, where the top goal *C1* is

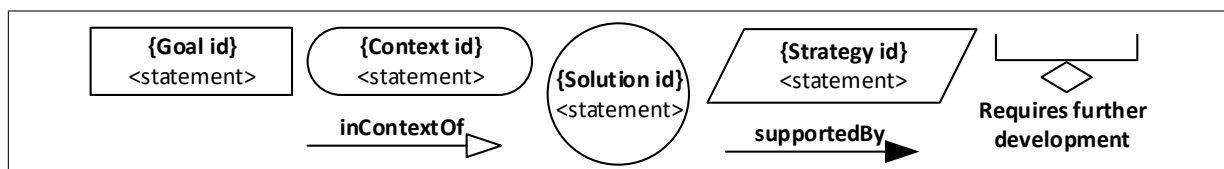


Figure E.1: GSN Elements

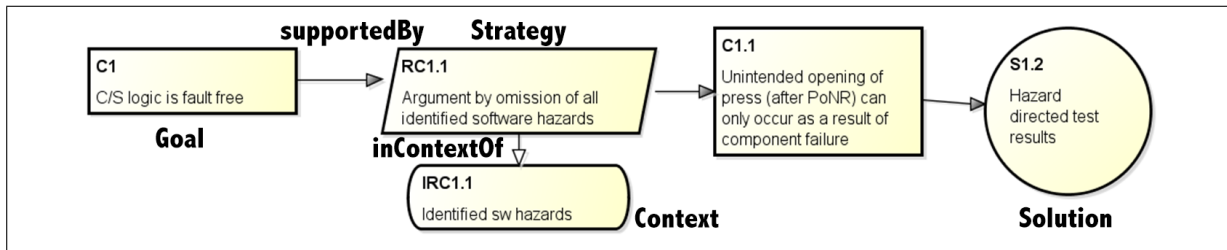


Figure E.2: An example of a GSN argument

decomposed via strategy *RC1.1*. The strategy is clarified by the context statement *IRC1.1*. The strategy is further supported by the goal *C1.1*, which is finally supported by the solution *S1.2*.

E.2 Machine-readable representation

To support argument portability between the different representations and tools, a standardised SACM/ARM XMI argument format [9] is available. SACM defines a meta-model for representing structured assurance cases which communicate the safety and security of a system. The safety of the system is represented with *Claims* (equivalent to Goals in GSN) that can be supported by reasoning [9]. The reasoning is done by associating claims together to support a bigger claim in an inferential way. Structured arguments usually make reference to evidences.

The importing and exporting of arguments in XMI format is supported by different tools (e.g., Astah GSN Editor¹) such that a graphical (or even a textual) argument can be stored and/or previewed using the same XMI format. The XMI format of the argument from Figure E.2 is shown in Figure E.3. Both GSN and XMI examples are adapted from the SACM standard document [9].

E.3 Textual representation

People have different learning patterns. While some tend to be more visual others still prefer text [42]. For that reason, Holloway [68] presented five textual representations of safety assurance cases. He uses a GSN example and represents it in these notations. The textual notations are:

¹<http://astah.net/editions/gsn>

Appendix E: Safety Case Representation

```
<ARM:Argumentation>
<argumentElement xsi:type="ARM:Claim" xmi:id="1" id="C1" content="C/S
  logic is fault free"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="2" id="RC1.1"
  content="Argument by omission of all identified software hazards"
  describedInference="16"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="4" id="IRC1.1
  " content="Identified sw hazards"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="5" id="C1.1" content="
  Unintended opening of press (after PoNR) can only occur as a result
  of component failure"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="11" id="S1.2"
  content="Hazard directed test results"/>
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="16" id="C1.1.1
  " source="5" target="1"/>
<argumentElement xsi:type="ARM:AssertedContext" xmi:id="21" id="CIRC1.1"
  source="4" target="2"/>
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="22" id="S1.1"
  source="11" target="5"/>
</ARM:Argumentation>
```

Figure E.3: The GSN argument encoded in SACM/ARM XMI format

We know that catastrophic hazard H2 has been sufficiently mitigated because fault tree analysis shows that its probability of occurrence is less than 1×10^{-6} per annum, and the acceptable probability in our environment for a catastrophic hazard is 1×10^{-6} per annum.

Figure E.4: Example of the normal prose [68]

- **Normal prose** is a normal textual representation of safety cases and is widely used in the law and philosophy fields. The problem with this notation is that it is very easy to lose of the structure of the argument between the words, and that is where the next representation comes in play. Figure E.4 shows an example of a normal prose,
- **Structured prose** to overcome the possible loss of structure in the normal prose, a structure can be added to the prose by explicitly denoting the critical parts of the safety case. Figure E.5 shows an example of structured prose,
- **Argument outline** for further structuring, numerical outlines can be used to represent the structure of the safety case argument. The text used is almost identical to the one used to annotate a GSN diagram. Figure E.6 shows an

Appendix E: Safety Case Representation

The evidence that catastrophic hazard H2 has been sufficiently mitigated is a fault tree analysis showing that its probability of occurrence is less than 1×10^{-6} per annum. The justification for using this evidence is that the acceptable probability in our environment for a catastrophic hazard is 1×10^{-6} per annum.

Figure E.5: Example of the structured prose [68]

Claim 1.1.2: Probability of H2 occurring $< 1 \times 10^{-6}$ per annum.
Justification 1.1.2: 1×10^{-6} per annum limit for catastrophic hazards.
Evidence 1.1.2.: Fault Tree analysis.

Figure E.6: Example of the argument outline [68]

```
...  
(claim H2 OK  
(justification CatHaz)  
(evidence FTA))  
...
```

Figure E.7: Example of the LISP style [68]

example of the argument outline,

- **Mathematical proof** this is inspired from geometry proofs where statements are supported by reasons which are either given assumptions or reference to statements established later in the proof,
- **Lisp style** this format is based on the programming language *LISP*. Figure E.7 shows an example of this format.

F

THE BSCU *Flamm* ARCHITECTURAL MODEL

```
<?xml version="1.0" encoding="ASCII"?>
<flamm:CompositeComponent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:flamm="
  http://www.polarsys.org/chess/fla/flamm" id="model::modelComponentView::bscuSys" name="
  bscuSys">
  <inputPorts id="model::modelComponentView::bscuSys::pedal2" name="pedal2" connectedPorts="//
    @components.2/@inputPorts.1 //@components.1/@inputPorts.0" owner="//>
    <failures id="noFailure"/>
    <failures type="failure" id="late"/>
  </inputPorts>
  <inputPorts id="model::modelComponentView::bscuSys::pedal1" name="pedal1" connectedPorts="//
    @components.1/@inputPorts.1 //@components.2/@inputPorts.0" owner="//>
    <failures id="noFailure"/>
  </inputPorts>
  <outputPorts id="model::modelComponentView::bscuSys::cmd" name="cmd" connectedPorts="//
    @components.0/@outputPorts.0" owner="//>
  <outputPorts id="model::modelComponentView::bscuSys::valid" name="valid" connectedPorts="//
    @components.3/@outputPorts.0" owner="//>
  <components xsi:type="flamm:SimpleComponent" id="
    model::modelComponentView::bscuSys::selectSwitch" name="selectSwitch">
    <inputPorts id="model::modelComponentView::selectSwitchImpl::valid" name="valid"
      connectedPorts="//@components.1/@outputPorts.0" owner="//@components.0"/>
    <inputPorts id="model::modelComponentView::selectSwitchImpl::cmd2" name="cmd2"
      connectedPorts="//@components.2/@outputPorts.1" owner="//@components.0"/>
    <inputPorts id="model::modelComponentView::selectSwitchImpl::cmd1" name="cmd1"
      connectedPorts="//@components.1/@outputPorts.1" owner="//@components.0"/>
    <outputPorts id="model::modelComponentView::selectSwitchImpl::cmd" name="cmd"
      connectedPorts="//@outputPorts.0" owner="//@components.0"/>
    <rules>
      <inputExpression port="//@components.0/@inputPorts.0">
        <failures id="noFailure"/>
      </inputExpression>
      <inputExpression port="//@components.0/@inputPorts.2">
        <failures type="failure" id="late"/>
      </inputExpression>
      <inputExpression port="//@components.0/@inputPorts.1">
        <failures type="failure" id="late"/>
      </inputExpression>
      <outputExpression port="//@components.0/@outputPorts.0">
        <failures type="failure" id="late"/>
      </outputExpression>
    </rules>
  </rules>
  <rules>
    <inputExpression port="//@components.0/@inputPorts.0">
      <failures id="noFailure"/>
    </inputExpression>
    <inputExpression port="//@components.0/@inputPorts.2">
      <failures type="failure" id="omission"/>
    </inputExpression>
    <inputExpression port="//@components.0/@inputPorts.1">
      <failures type="failure" id="omission"/>
    </inputExpression>
    <outputExpression port="//@components.0/@outputPorts.0">
      <failures type="failure" id="omission"/>
    </outputExpression>
  </rules>
</components>
<components xsi:type="flamm:SimpleComponent" id="
  model::modelComponentView::bscuSys::subBSCU1" name="subBSCU1">
```

Appendix F: The BSCU *Flamm* Architectural Model

```
<inputPorts id="model::modelComponentView::subBSCU1impl::pedal2" name="pedal2"
connectedPorts="//@inputPorts.0" owner="//@components.1"/>
<inputPorts id="model::modelComponentView::subBSCU1impl::pedal1" name="pedal1"
connectedPorts="//@inputPorts.1" owner="//@components.1"/>
<outputPorts id="model::modelComponentView::subBSCU1impl::valid" name="valid"
connectedPorts="//@components.3/@inputPorts.1 //@components.0/@inputPorts.0" owner="//
@components.1"/>
<outputPorts id="model::modelComponentView::subBSCU1impl::cmd" name="cmd" connectedPorts="//
@components.0/@inputPorts.2" owner="//@components.1"/>
<rules>
  <inputExpression port="//@components.1/@inputPorts.1">
    <failures type="failure" id="late"/>
  </inputExpression>
  <inputExpression port="//@components.1/@inputPorts.0">
    <failures type="failure" id="late"/>
  </inputExpression>
  <outputExpression port="//@components.1/@outputPorts.0">
    <failures type="failure" id="late"/>
  </outputExpression>
  <outputExpression port="//@components.1/@outputPorts.1">
    <failures type="failure" id="late"/>
  </outputExpression>
</rules>
<rules>
  <inputExpression port="//@components.1/@inputPorts.1">
    <failures type="failure" id="late"/>
  </inputExpression>
  <inputExpression port="//@components.1/@inputPorts.0">
    <failures id="noFailure"/>
  </inputExpression>
  <outputExpression port="//@components.1/@outputPorts.0">
    <failures id="noFailure"/>
  </outputExpression>
  <outputExpression port="//@components.1/@outputPorts.1">
    <failures type="failure" id="omission"/>
  </outputExpression>
</rules>
<rules>
  <inputExpression port="//@components.1/@inputPorts.1">
    <failures id="noFailure"/>
  </inputExpression>
  <inputExpression port="//@components.1/@inputPorts.0">
    <failures type="failure" id="late"/>
  </inputExpression>
  <outputExpression port="//@components.1/@outputPorts.0">
    <failures id="noFailure"/>
  </outputExpression>
  <outputExpression port="//@components.1/@outputPorts.1">
    <failures type="failure" id="omission"/>
  </outputExpression>
</rules>
</components>
<components xsi:type="flamm:SimpleComponent" id="
model::modelComponentView::bscuSys::subBSCU2" name="subBSCU2">
  <inputPorts id="model::modelComponentView::subBSCU2impl::pedal1" name="pedal1"
connectedPorts="//@inputPorts.1" owner="//@components.2"/>
  <inputPorts id="model::modelComponentView::subBSCU2impl::pedal2" name="pedal2"
connectedPorts="//@inputPorts.0" owner="//@components.2"/>
  <outputPorts id="model::modelComponentView::subBSCU2impl::valid" name="valid"
connectedPorts="//@components.3/@inputPorts.0" owner="//@components.2"/>
  <outputPorts id="model::modelComponentView::subBSCU2impl::cmd" name="cmd" connectedPorts="//
@components.0/@inputPorts.1" owner="//@components.2"/>
  <rules>
    <inputExpression port="//@components.2/@inputPorts.0">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.2/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
    <outputExpression port="//@components.2/@outputPorts.0">
      <failures type="failure" id="late"/>
    </outputExpression>
    <outputExpression port="//@components.2/@outputPorts.1">
      <failures type="failure" id="late"/>
    </outputExpression>
  </rules>
</components>
```

Appendix F: The BSCU *Flamm* Architectural Model

```
</outputExpression>
</rules>
<rules>
  <inputExpression port="//@components.2/@inputPorts.0">
    <failures type="failure" id="late"/>
  </inputExpression>
  <inputExpression port="//@components.2/@inputPorts.1">
    <failures id="noFailure"/>
  </inputExpression>
  <outputExpression port="//@components.2/@outputPorts.0">
    <failures id="noFailure"/>
  </outputExpression>
  <outputExpression port="//@components.2/@outputPorts.1">
    <failures type="failure" id="omission"/>
  </outputExpression>
</rules>
<rules>
  <inputExpression port="//@components.2/@inputPorts.0">
    <failures id="noFailure"/>
  </inputExpression>
  <inputExpression port="//@components.2/@inputPorts.1">
    <failures type="failure" id="late"/>
  </inputExpression>
  <outputExpression port="//@components.2/@outputPorts.0">
    <failures id="noFailure"/>
  </outputExpression>
  <outputExpression port="//@components.2/@outputPorts.1">
    <failures type="failure" id="omission"/>
  </outputExpression>
</rules>
</components>
<components xsi:type="flamm:SimpleComponent" id="
  model::modelComponentView::bscuSys::validSwitch" name="validSwitch">
  <inputPorts id="model::modelComponentView::validSwitchImpl::valid2" name="valid2"
  connectedPorts="//@components.2/@outputPorts.0" owner="//@components.3"/>
  <inputPorts id="model::modelComponentView::validSwitchImpl::valid1" name="valid1"
  connectedPorts="//@components.1/@outputPorts.0" owner="//@components.3"/>
  <outputPorts id="model::modelComponentView::validSwitchImpl::valid" name="valid"
  connectedPorts="//@outputPorts.1" owner="//@components.3"/>
  <rules>
    <inputExpression port="//@components.3/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.3/@inputPorts.0">
      <failures type="failure" id="late"/>
    </inputExpression>
    <outputExpression port="//@components.3/@outputPorts.0">
      <failures type="failure" id="late"/>
    </outputExpression>
  </rules>
  <rules>
    <inputExpression port="//@components.3/@inputPorts.1">
      <failures id="noFailure"/>
    </inputExpression>
    <inputExpression port="//@components.3/@inputPorts.0">
      <failures type="failure" id="late"/>
    </inputExpression>
    <outputExpression port="//@components.3/@outputPorts.0">
      <failures id="noFailure"/>
    </outputExpression>
  </rules>
  <rules>
    <inputExpression port="//@components.3/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.3/@inputPorts.0">
      <failures id="noFailure"/>
    </inputExpression>
    <outputExpression port="//@components.3/@outputPorts.0">
      <failures id="noFailure"/>
    </outputExpression>
  </rules>
</components>
</flamm:CompositeComponent>
```



THE BSCU *Flamm* ARCHITECTURAL MODEL WITH FPTC RESULTS

```
<?xml version="1.0" encoding="ASCII"?>
<flamm:CompositeComponent xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="
  http://www.w3.org/2001/XMLSchema-instance" xmlns:flamm="http://www.polarsys.org/chess/fla/
  flamm" id="model::modelComponentView::bscuSys" name="bscuSys">
  <inputPorts id="model::modelComponentView::bscuSys::pedal2" name="pedal2" connectedPorts="//
    @components.2/@inputPorts.1 // @components.1/@inputPorts.0" owner="/">
    <failures id="noFailure"/>
    <failures type="failure" id="late"/>
  </inputPorts>
  <inputPorts id="model::modelComponentView::bscuSys::pedal1" name="pedal1" connectedPorts="//
    @components.1/@inputPorts.1 // @components.2/@inputPorts.0" owner="/">
    <failures id="noFailure"/>
  </inputPorts>
  <outputPorts id="model::modelComponentView::bscuSys::cmd" name="cmd" connectedPorts="//
    @components.0/@outputPorts.0" owner="/">
    <failures id="noFailure" previousFailures="//@components.0/@outputPorts.0/@failures.0"/>
    <failures type="failure" id="omission" previousFailures="//@components.0/@outputPorts.0/
    @failures.1"/>
  </outputPorts>
  <outputPorts id="model::modelComponentView::bscuSys::valid" name="valid" connectedPorts="//
    @components.3/@outputPorts.0" owner="/">
    <failures id="noFailure" previousFailures="//@components.3/@outputPorts.0/@failures.0"/>
  </outputPorts>
  <components xsi:type="flamm:SimpleComponent" id="
    model::modelComponentView::bscuSys::selectSwitch" name="selectSwitch">
    <inputPorts id="model::modelComponentView::selectSwitchImpl::valid" name="valid"
      connectedPorts="//@components.1/@outputPorts.0" owner="//@components.0">
      <failures id="noFailure" previousFailures="//@components.1/@outputPorts.0/@failures.0"/>
    </inputPorts>
    <inputPorts id="model::modelComponentView::selectSwitchImpl::cmd2" name="cmd2"
      connectedPorts="//@components.2/@outputPorts.1" owner="//@components.0">
      <failures id="noFailure" previousFailures="//@components.2/@outputPorts.1/@failures.0"/>
      <failures type="failure" id="omission" previousFailures="//@components.2/@outputPorts.1/
      @failures.1"/>
    </inputPorts>
    <inputPorts id="model::modelComponentView::selectSwitchImpl::cmd1" name="cmd1"
      connectedPorts="//@components.1/@outputPorts.1" owner="//@components.0">
      <failures id="noFailure" previousFailures="//@components.1/@outputPorts.1/@failures.0"/>
      <failures type="failure" id="omission" previousFailures="//@components.1/@outputPorts.1/
      @failures.1"/>
    </inputPorts>
    <outputPorts id="model::modelComponentView::selectSwitchImpl::cmd" name="cmd"
      connectedPorts="//@outputPorts.0" owner="//@components.0">
      <failures id="noFailure" previousFailures="//@components.0/@inputPorts.0/@failures.0 //
      @components.0/@inputPorts.1/@failures.0 // @components.0/@inputPorts.2/@failures.0"/>
      <failures type="failure" id="omission" previousFailures="//@components.0/@inputPorts.1/
      @failures.1 // @components.0/@inputPorts.2/@failures.1 // @components.0/@inputPorts.0/
      @failures.0"/>
    </outputPorts>
  </rules>
  <inputExpression port="//@components.0/@inputPorts.0">
    <failures id="noFailure"/>
  </inputExpression>
  <inputExpression port="//@components.0/@inputPorts.2">
```

Appendix G: The BSCU *Flamm* Architectural Model With FPTC Results

```
<failures type="failure" id="late"/>
</inputExpression>
<inputExpression port="//@components.0/@inputPorts.1">
  <failures type="failure" id="late"/>
</inputExpression>
<outputExpression port="//@components.0/@outputPorts.0">
  <failures type="failure" id="late"/>
</outputExpression>
</rules>
<rules>
  <inputExpression port="//@components.0/@inputPorts.0">
    <failures id="noFailure"/>
  </inputExpression>
  <inputExpression port="//@components.0/@inputPorts.2">
    <failures type="failure" id="omission"/>
  </inputExpression>
  <inputExpression port="//@components.0/@inputPorts.1">
    <failures type="failure" id="omission"/>
  </inputExpression>
  <outputExpression port="//@components.0/@outputPorts.0">
    <failures type="failure" id="omission"/>
  </outputExpression>
</rules>
</components>
<components xsi:type="flamm:SimpleComponent" id="
  model::modelComponentView::bscuSys::subBSCU1" name="subBSCU1">
  <inputPorts id="model::modelComponentView::subBSCU1impl::pedal2" name="pedal2"
  connectedPorts="//@inputPorts.0" owner="//@components.1">
    <failures id="noFailure" previousFailures="//@inputPorts.0/@failures.0"/>
    <failures type="failure" id="late" previousFailures="//@inputPorts.0/@failures.1"/>
  </inputPorts>
  <inputPorts id="model::modelComponentView::subBSCU1impl::pedal1" name="pedal1"
  connectedPorts="//@inputPorts.1" owner="//@components.1">
    <failures id="noFailure" previousFailures="//@inputPorts.1/@failures.0"/>
  </inputPorts>
  <outputPorts id="model::modelComponentView::subBSCU1impl::valid" name="valid"
  connectedPorts="//@components.3/@inputPorts.1 //@components.0/@inputPorts.0" owner="//
  @components.1">
    <failures id="noFailure" previousFailures="//@components.1/@inputPorts.0/@failures.1 //
  @components.1/@inputPorts.1/@failures.0 //@components.1/@inputPorts.0/@failures.0"/>
  </outputPorts>
  <outputPorts id="model::modelComponentView::subBSCU1impl::cmd" name="cmd" connectedPorts="
  //@components.0/@inputPorts.2" owner="//@components.1">
    <failures id="noFailure" previousFailures="//@components.1/@inputPorts.0/@failures.0 //
  @components.1/@inputPorts.1/@failures.0"/>
    <failures type="failure" id="omission" previousFailures="//@components.1/@inputPorts.0/
  @failures.1 //@components.1/@inputPorts.1/@failures.0"/>
  </outputPorts>
  <rules>
    <inputExpression port="//@components.1/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.1/@inputPorts.0">
      <failures type="failure" id="late"/>
    </inputExpression>
    <outputExpression port="//@components.1/@outputPorts.0">
      <failures type="failure" id="late"/>
    </outputExpression>
    <outputExpression port="//@components.1/@outputPorts.1">
      <failures type="failure" id="late"/>
    </outputExpression>
  </rules>
  <rules>
    <inputExpression port="//@components.1/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.1/@inputPorts.0">
      <failures id="noFailure"/>
    </inputExpression>
    <outputExpression port="//@components.1/@outputPorts.0">
      <failures id="noFailure"/>
    </outputExpression>
    <outputExpression port="//@components.1/@outputPorts.1">
      <failures type="failure" id="omission"/>
    </outputExpression>
  </rules>
</components>
```

Appendix G: The BSCU *Flamm* Architectural Model With FPTC Results

```
</outputExpression>
</rules>
<rules>
  <inputExpression port="//@components.1/@inputPorts.1">
    <failures id="noFailure"/>
  </inputExpression>
  <inputExpression port="//@components.1/@inputPorts.0">
    <failures type="failure" id="late"/>
  </inputExpression>
  <outputExpression port="//@components.1/@outputPorts.0">
    <failures id="noFailure"/>
  </outputExpression>
  <outputExpression port="//@components.1/@outputPorts.1">
    <failures type="failure" id="omission"/>
  </outputExpression>
</rules>
</components>
<components xsi:type="flamm:SimpleComponent" id="
  model::modelComponentView::bscuSys::subBSCU2" name="subBSCU2">
  <inputPorts id="model::modelComponentView::subBSCU2impl::pedal1" name="pedal1"
  connectedPorts="//@inputPorts.1" owner="//@components.2">
    <failures id="noFailure" previousFailures="//@inputPorts.1/@failures.0"/>
  </inputPorts>
  <inputPorts id="model::modelComponentView::subBSCU2impl::pedal2" name="pedal2"
  connectedPorts="//@inputPorts.0" owner="//@components.2">
    <failures id="noFailure" previousFailures="//@inputPorts.0/@failures.0"/>
    <failures type="failure" id="late" previousFailures="//@inputPorts.0/@failures.1"/>
  </inputPorts>
  <outputPorts id="model::modelComponentView::subBSCU2impl::valid" name="valid"
  connectedPorts="//@components.3/@inputPorts.0" owner="//@components.2">
    <failures id="noFailure" previousFailures="//@components.2/@inputPorts.0/@failures.0 //
  @components.2/@inputPorts.1/@failures.0 //@components.2/@inputPorts.1/@failures.1"/>
  </outputPorts>
  <outputPorts id="model::modelComponentView::subBSCU2impl::cmd" name="cmd" connectedPorts=
  //@components.0/@inputPorts.1" owner="//@components.2">
    <failures id="noFailure" previousFailures="//@components.2/@inputPorts.0/@failures.0 //
  @components.2/@inputPorts.1/@failures.0"/>
    <failures type="failure" id="omission" previousFailures="//@components.2/@inputPorts.0/
  @failures.0 //@components.2/@inputPorts.1/@failures.1"/>
  </outputPorts>
  <rules>
    <inputExpression port="//@components.2/@inputPorts.0">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.2/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
    <outputExpression port="//@components.2/@outputPorts.0">
      <failures type="failure" id="late"/>
    </outputExpression>
    <outputExpression port="//@components.2/@outputPorts.1">
      <failures type="failure" id="late"/>
    </outputExpression>
  </rules>
  <rules>
    <inputExpression port="//@components.2/@inputPorts.0">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.2/@inputPorts.1">
      <failures id="noFailure"/>
    </inputExpression>
    <outputExpression port="//@components.2/@outputPorts.0">
      <failures id="noFailure"/>
    </outputExpression>
    <outputExpression port="//@components.2/@outputPorts.1">
      <failures type="failure" id="omission"/>
    </outputExpression>
  </rules>
  <rules>
    <inputExpression port="//@components.2/@inputPorts.0">
      <failures id="noFailure"/>
    </inputExpression>
    <inputExpression port="//@components.2/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
  </rules>
</components>
```

Appendix G: The BSCU *Flamm* Architectural Model With FPTC Results

```
</inputExpression>
<outputExpression port="//@components.2/@outputPorts.0">
  <failures id="noFailure"/>
</outputExpression>
<outputExpression port="//@components.2/@outputPorts.1">
  <failures type="failure" id="omission"/>
</outputExpression>
</rules>
</components>
<components xsi:type="flamm:SimpleComponent" id="
model::modelComponentView::bscuSys::validSwitch" name="validSwitch">
  <inputPorts id="model::modelComponentView::validSwitchImpl::valid2" name="valid2"
connectedPorts="//@components.2/@outputPorts.0" owner="//@components.3">
  <failures id="noFailure" previousFailures="//@components.2/@outputPorts.0/@failures.0"/>
</inputPorts>
  <inputPorts id="model::modelComponentView::validSwitchImpl::valid1" name="valid1"
connectedPorts="//@components.1/@outputPorts.0" owner="//@components.3">
  <failures id="noFailure" previousFailures="//@components.1/@outputPorts.0/@failures.0"/>
</inputPorts>
  <outputPorts id="model::modelComponentView::validSwitchImpl::valid" name="valid"
connectedPorts="//@outputPorts.1" owner="//@components.3">
  <failures id="noFailure" previousFailures="//@components.3/@inputPorts.0/@failures.0 //
@components.3/@inputPorts.1/@failures.0"/>
</outputPorts>
  <rules>
    <inputExpression port="//@components.3/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.3/@inputPorts.0">
      <failures type="failure" id="late"/>
    </inputExpression>
    <outputExpression port="//@components.3/@outputPorts.0">
      <failures type="failure" id="late"/>
    </outputExpression>
  </rules>
  <rules>
    <inputExpression port="//@components.3/@inputPorts.1">
      <failures id="noFailure"/>
    </inputExpression>
    <inputExpression port="//@components.3/@inputPorts.0">
      <failures type="failure" id="late"/>
    </inputExpression>
    <outputExpression port="//@components.3/@outputPorts.0">
      <failures id="noFailure"/>
    </outputExpression>
  </rules>
  <rules>
    <inputExpression port="//@components.3/@inputPorts.1">
      <failures type="failure" id="late"/>
    </inputExpression>
    <inputExpression port="//@components.3/@inputPorts.0">
      <failures id="noFailure"/>
    </inputExpression>
    <outputExpression port="//@components.3/@outputPorts.0">
      <failures id="noFailure"/>
    </outputExpression>
  </rules>
</components>
</flamm:CompositeComponent>
```

H

THE UNDESIRED HAZARDOUS EVENTS

```
<parameters>
  <domain>ARP4754A</domain>

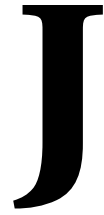
  <criticalityLevels>
    <!-- change the criticality expressions below according
    to the desired standard/domain. You can add more levels
    as necessary. -->
    <level id="1" expression="negligible" />
    <!-- lowest criticality (negligible) level -->
    <level id="2" expression="minor" />
    <level id="3" expression="major" />
    <level id="4" expression="hazardous" />
    <level id="5" expression="catastrophic" />
    <!-- highest criticality level -->
  </criticalityLevels>

  <hazardousEvents>
    <failure type="omission" criticality="5"/>
    <failure type="late" criticality="2"/>
  </hazardousEvents>
</parameters>
```




THE SACM/XMI REPRESENTATION OF THE PRODUCT-BASED ARGUMENT

```
<?xml version="1.0" encoding="utf-8"?>
<ARM:Argumentation xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:ARM="www.omg.org/spec/SACM/20120501/
Argumentation" xmi:id="0" id="GSN">
<argumentElement xsi:type="ARM:Claim" xmi:id="1" id="C1" content="All causes of hazardous
Failure Modes are acceptable" toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="2" id="C1.1" content="Hazardous Failure Mode of
type &apos;late&apos; is absent in contributory software functionality." toBeSupported="
false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="3" id="IC1.1" content="Known causes
of late failure mode." toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="5" id="RC1.1" content="Argument over
failure mechanisms " describedInference="17 18 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="6" id="IRC1.1" content="Identified
failure mechanisms describe all known causes of Late hazardous Failure Mode" toBeSupported
="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="8" id="C1.1.1" content="The known causes of
secondary failures of other components are acceptably handled" toBeSupported="true"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="9" id="C1.1.2" content="The component
successfully handles the primary failures" toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="10" id="C1.1.2.1" content="Hazardous event: late
has been mitigated by component(s): subBSCU1,subBSCU2" toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="11" id="S1.1.2.1" content="
Mitigation details in product_arg.txt." toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="12" id="C1.2" content="Hazardous Failure Mode of
type &apos;omission&apos; is absent in contributory software functionality."
toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="13" id="IC1.2" content="Known
causes of omission failure mode." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="15" id="S1.2" content="Counter
Evidence in product_arg.txt." toBeSupported="false"/>
<argumentElement xsi:type="ARM:AssertedContext" xmi:id="4" id="C1.1" toBeSupported="false"
source="3" target="2"/>
<argumentElement xsi:type="ARM:AssertedContext" xmi:id="7" id="RC1.1" toBeSupported="false"
source="6" target="5"/>
<argumentElement xsi:type="ARM:AssertedContext" xmi:id="14" id="C1.2" toBeSupported="false"
source="13" target="12"/>
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="16" id="C1.1" toBeSupported="false"
source="2" target="1" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="17" id="C1.1.1" toBeSupported="false
" source="8" target="2" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="18" id="C1.1.2" toBeSupported="false
" source="9" target="2" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="19" id="C1.1.2.1" toBeSupported="
false" source="10" target="9" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="20" id="S1.1.2.1" toBeSupported="
false" source="11" target="10" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="21" id="C1.2" toBeSupported="false"
source="12" target="1" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="22" id="S1.2" toBeSupported="false"
source="15" target="12" /></ARM:Argumentation>
```



THE ARGUMENT OUTLINE TEXTUAL REPRESENTATION OF THE PRODUCT-BASED ARGUMENT

CLAIM C1: ALL CAUSES OF HAZARDOUS FAILURE MODES ARE ACCEPTABLE

CLAIM C1.1: HAZARDOUS FAILURE MODE OF TYPE 'LATE' IS ABSENT IN CONTRIBUTORY SOFTWARE FUNCTIONALITY.

CONTEXT C1.1: Known causes of Late failure mode.

STRATEGY RC1.1: Argument over failure mechanisms

JUSTIFICATION RC1.1: Identified failure mechanisms describe all known causes of Late hazardous Failure Mode

CLAIM C1.1.1: THE KNOWN CAUSES OF SECONDARY FAILURES OF OTHER COMPONENTS ARE ACCEPTABLY HANDLED
[Undeveloped]

CLAIM C1.1.2: THE COMPONENT SUCCESSFULLY HANDLES THE PRIMARY FAILURES

CLAIM C1.1.2.1: HAZARDOUS EVENT: LATE HAS BEEN MITIGATED BY COMPONENT(S): SUBBSCU1, SUBBSCU2

CONTEXT C1.1.2.1: late is mitigated by: subBSCU1

through rule: RULE:

INPUT EXPRESSION: -PORT: //@components.1/@inputPorts.1

Failure: -TYPE: failure -ID: late -PREVIOUS_FAILURES: null

INPUT EXPRESSION: -PORT: //@components.1/@inputPorts.0

Failure: -TYPE: null -ID: noFailure -PREVIOUS_FAILURES: null

OUTPUT EXPRESSION: -PORT: //@components.1/@outputPorts.0

Failure: -TYPE: null -ID: noFailure -PREVIOUS_FAILURES: null

OUTPUT EXPRESSION: -PORT: //@components.1/@outputPorts.1

Failure: -TYPE: failure -ID: omission -PREVIOUS_FAILURES: null

through rule: RULE:

INPUT EXPRESSION: -PORT: //@components.1/@inputPorts.1

Failure: -TYPE: null -ID: noFailure -PREVIOUS_FAILURES: null

INPUT EXPRESSION: -PORT: //@components.1/@inputPorts.0

Failure: -TYPE: failure -ID: late -PREVIOUS_FAILURES: null

OUTPUT EXPRESSION: -PORT: //@components.1/@outputPorts.0

Failure: -TYPE: null -ID: noFailure -PREVIOUS_FAILURES: null

OUTPUT EXPRESSION: -PORT: //@components.1/@outputPorts.1

Failure: -TYPE: failure -ID: omission -PREVIOUS_FAILURES: null

Late is mitigated by: subBSCU2

through rule: RULE:

Appendix J: The Argument Outline Textual Representation of the Product-Based Argument

```
INPUT EXPRESSION: -PORT: //@components.2/@inputPorts.0
Failure: -TYPE: failure -ID: late -PREVIOUS_FAILURES: null

INPUT EXPRESSION: -PORT: //@components.2/@inputPorts.1
Failure: -TYPE: null -ID: noFailure -PREVIOUS_FAILURES: null

OUTPUT EXPRESSION: -PORT: //@components.2/@outputPorts.0
Failure: -TYPE: null -ID: noFailure -PREVIOUS_FAILURES: null

OUTPUT EXPRESSION: -PORT: //@components.2/@outputPorts.1
Failure: -TYPE: failure -ID: omission -PREVIOUS_FAILURES: null

through rule: RULE:
INPUT EXPRESSION: -PORT: //@components.2/@inputPorts.0
Failure: -TYPE: null -ID: noFailure -PREVIOUS_FAILURES: null

INPUT EXPRESSION: -PORT: //@components.2/@inputPorts.1
Failure: -TYPE: failure -ID: late -PREVIOUS_FAILURES: null

OUTPUT EXPRESSION: -PORT: //@components.2/@outputPorts.0
Failure: -TYPE: null -ID: noFailure -PREVIOUS_FAILURES: null

OUTPUT EXPRESSION: -PORT: //@components.2/@outputPorts.1
Failure: -TYPE: failure -ID: omission -PREVIOUS_FAILURES: null

EVIDENCE S1.1.2.1: Mitigation details in product_arg.txt.
```

CLAIM C1.2: HAZARDOUS FAILURE MODE OF TYPE 'OMISSION' IS ABSENT IN CONTRIBUTORY SOFTWARE

FUNCTIONALITY.

CONTEXT C1.2: Known causes of omission failure mode.

COUNTER_EVIDENCE S1.2: Counter Evidence in product_arg.txt.

CONTEXT S1.2: Omission CAUSED BY:

```
Failure: 'omission' On Output Port: 'cmd' of Component: 'selectSwitch'. CAUSED BY:
{
Failure: 'omission' On Input Port: 'cmd2' of Component: 'selectSwitch'. CAUSED BY:
Failure: 'omission' On Output Port: 'cmd' of Component: 'subBSCU2'. CAUSED BY:
{
Failure: 'noFailure' On Input Port: 'pedal1' of Component: 'subBSCU2'. CAUSED BY:
Failure: 'noFailure' On Input Port: 'pedal1' of Component: 'System [the composite
component]'. CAUSED BY: NO FURTHER CAUSES.
```

AND

```
Failure: 'late' On Input Port: 'pedal2' of Component: 'subBSCU2'. CAUSED BY:
Failure: 'late' On Input Port: 'pedal2' of Component: 'System [the composite component
]'.
CAUSED BY: NO FURTHER CAUSES.
```

}

AND

```
Failure: 'omission' On Input Port: 'cmd1' of Component: 'selectSwitch'. CAUSED BY:
Failure: 'omission' On Output Port: 'cmd' of Component: 'subBSCU1'. CAUSED BY:
{
Failure: 'late' On Input Port: 'pedal2' of Component: 'subBSCU1'. CAUSED BY:
Failure: 'late' On Input Port: 'pedal2' of Component: 'System [the composite component
]'.
CAUSED BY: NO FURTHER CAUSES.
```

AND

```
Failure: 'noFailure' On Input Port: 'pedal1' of Component: 'subBSCU1'. CAUSED BY:
Failure: 'noFailure' On Input Port: 'pedal1' of Component: 'System [the composite
component]'. CAUSED BY: NO FURTHER CAUSES.
```

}

AND

Appendix J: The Argument Outline Textual Representation of the Product-Based Argument

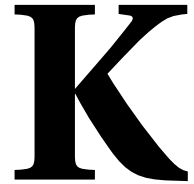
```
Failure: 'noFailure' On Input Port: 'valid' of Component: 'selectSwitch'. CAUSED BY:
Failure: 'noFailure' On Output Port: 'valid' of Component: 'subBSCU1'. CAUSED BY:
{
Failure: 'late' On Input Port: 'pedal2' of Component: 'subBSCU1'. CAUSED BY:
Failure: 'late' On Input Port: 'pedal2' of Component: 'System [the composite component
]'.
CAUSED BY: NO FURTHER CAUSES.

AND
Failure: 'noFailure' On Input Port: 'pedal1' of Component: 'subBSCU1'. CAUSED BY:
Failure: 'noFailure' On Input Port: 'pedal1' of Component: 'System [the composite
component]'. CAUSED BY: NO FURTHER CAUSES.

AND
Failure: 'noFailure' On Input Port: 'pedal2' of Component: 'subBSCU1'. CAUSED BY:
Failure: 'noFailure' On Input Port: 'pedal2' of Component: 'System [the composite
component]'. CAUSED BY: NO FURTHER CAUSES.

}

}
```



THE SACM/XMI REPRESENTATION OF THE PROCESS-BASED ARGUMENT

```
<?xml version="1.0" encoding="utf-8"?>
<ARM:Argumentation xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance" xmlns:ARM="www.omg.org/spec/SACM/20120501/
  Argumentation" xmi:id="0" id="GSN">
<argumentElement xsi:type="ARM:Claim" xmi:id="1" id="C1" content="The process meets the safety
  requirements." toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="2" id="C1.1" content="The activity Process-
  Based_Argument_Generation has been carried out." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="3" id="IC1.1" content="Standard:
  ARP4761 -- Performed @ 09/03/2016 18:21:21" toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="5" id="RC1.1" content="Argument over
  roles " describedInference="67 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="6" id="C1.1.1" content="Role Safety Engineer is
  certified." toBeSupported="true"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="8" id="WC1.1" content="Argument over
  work products " describedInference="69 71 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="9" id="C1.1.2" content="Process-Based SACM
  Argument is available." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="10" id="S1.1.2" content="
  processBasedSafetySACMArgument.xmi" toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="11" id="C1.1.3" content="Textual Argument is
  available." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="12" id="S1.1.3" content="
  processBasedSafetyArgument.txt" toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="13" id="TC1.1" content="Argument
  over tools " describedInference="73 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="14" id="C1.1.4" content="Tool Process-
  Based_Argument_Generation -version: 1 is qualified." toBeSupported="true"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="16" id="GC1.1" content="Argument
  over guidance " describedInference="75 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="17" id="C1.1.5" content="Guidance Appendix B3
  has been followed." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="18" id="S1.1.5" content="Appendix
  B3" toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="19" id="C1.2" content="The activity
  Arguments_Composition has been carried out." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="20" id="IC1.2" content="Standard:
  ARP4761 -- Performed @ 09/03/2016 18:21:21" toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="22" id="RC1.2" content="Argument
  over roles " describedInference="78 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="23" id="C1.2.6" content="Role Safety Engineer is
  certified." toBeSupported="true"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="25" id="WC1.2" content="Argument
  over work products " describedInference="80 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="26" id="C1.2.7" content="Safety Case Argument
  Fragment is available." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="27" id="S1.2.7" content="
  CombinedSafetySACMArgument.xmi" toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="28" id="TC1.2" content="Argument
  over tools " describedInference="82 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="29" id="C1.2.8" content="Tool
  Arguments_Composition -version: 1 is qualified." toBeSupported="true"/>
```

Appendix K: The SACM/XMI Representation of the Process-Based Argument

```
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="31" id="GC1.2" content="Argument
over guidance " describedInference="84 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="32" id="C1.2.9" content="Guidance Appendix B3
has been followed." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="33" id="S1.2.9" content="Appendix
B3" toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="34" id="C1.3" content="The activity FPTC_based
Analysis has been carried out." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="35" id="IC1.3" content="Standard:
ARP4761 -- Performed @ 09/03/2016 18:21:21" toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="37" id="RC1.3" content="Argument
over roles " describedInference="87 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="38" id="C1.3.10" content="Role Safety Engineer
is certified." toBeSupported="true"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="40" id="WC1.3" content="Argument
over work products " describedInference="89 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="41" id="C1.3.11" content="FPTC results is
available." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="42" id="S1.3.11" content="bscu.
flamm" toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="43" id="TC1.3" content="Argument
over tools " describedInference="91 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="44" id="C1.3.12" content="Tool FPTC_based
Analysis -version: 1 is qualified." toBeSupported="true"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="46" id="GC1.3" content="Argument
over guidance " describedInference="93 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="47" id="C1.3.13" content="Guidance Appendix B3
has been followed." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="48" id="S1.3.13" content="Appendix
B3" toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="49" id="C1.4" content="The activity Product-
Based_Argument_Generation has been carried out." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="50" id="IC1.4" content="Standard:
ARP4761 -- Performed @ 09/03/2016 18:21:21" toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="52" id="RC1.4" content="Argument
over roles " describedInference="96 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="53" id="C1.4.14" content="Role Safety Engineer
is certified." toBeSupported="true"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="55" id="WC1.4" content="Argument
over work products " describedInference="98 100 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="56" id="C1.4.15" content="Product-Based SACM
Argument is available." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="57" id="S1.4.15" content="
productBasedSafetySACMArgument.xmi" toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="58" id="C1.4.16" content="Textual Argument is
available." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="59" id="S1.4.16" content="
productBasedSafetyArgument.txt" toBeSupported="false"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="60" id="TC1.4" content="Argument
over tools " describedInference="102 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="61" id="C1.4.17" content="Tool Product-
Based_Argument_Generation -version: 3 is qualified." toBeSupported="true"/>
<argumentElement xsi:type="ARM:ArgumentReasoning" xmi:id="63" id="GC1.4" content="Argument
over guidance " describedInference="104 " toBeSupported="false"/>
<argumentElement xsi:type="ARM:Claim" xmi:id="64" id="C1.4.18" content="Guidance Appendix B3
has been followed." toBeSupported="false"/>
<argumentElement xsi:type="ARM:InformationElement" xmi:id="65" id="S1.4.18" content="Appendix
B3" toBeSupported="false"/>
<argumentElement xsi:type="ARM:AssertedContext" xmi:id="4" id="C1.1" toBeSupported="false"
source="3" target="2"/>
<argumentElement xsi:type="ARM:AssertedContext" xmi:id="21" id="C1.2" toBeSupported="false"
source="20" target="19"/>
<argumentElement xsi:type="ARM:AssertedContext" xmi:id="36" id="C1.3" toBeSupported="false"
source="35" target="34"/>
<argumentElement xsi:type="ARM:AssertedContext" xmi:id="51" id="C1.4" toBeSupported="false"
source="50" target="49"/>
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="66" id="C1.1" toBeSupported="false"
source="2" target="1" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="67" id="C1.1.1" toBeSupported="false"
" source="6" target="2" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="69" id="C1.1.2" toBeSupported="false"
" source="9" target="2" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="70" id="S1.1.2" toBeSupported="false"
source="10" target="9" />
```

Appendix K: The SACM/XMI Representation of the Process-Based Argument

```
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="71" id="C1.1.3" toBeSupported="false"
  " source="11" target="2" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="72" id="S1.1.3" toBeSupported="false"
  source="12" target="11" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="73" id="C1.1.4" toBeSupported="false"
  " source="14" target="2" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="75" id="C1.1.5" toBeSupported="false"
  " source="17" target="2" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="76" id="S1.1.5" toBeSupported="false"
  source="18" target="17" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="77" id="C1.2" toBeSupported="false"
  source="19" target="1" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="78" id="C1.2.6" toBeSupported="false"
  " source="23" target="19" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="80" id="C1.2.7" toBeSupported="false"
  " source="26" target="19" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="81" id="S1.2.7" toBeSupported="false"
  source="27" target="26" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="82" id="C1.2.8" toBeSupported="false"
  " source="29" target="19" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="84" id="C1.2.9" toBeSupported="false"
  " source="32" target="19" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="85" id="S1.2.9" toBeSupported="false"
  source="33" target="32" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="86" id="C1.3" toBeSupported="false"
  source="34" target="1" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="87" id="C1.3.10" toBeSupported="
  false" source="38" target="34" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="89" id="C1.3.11" toBeSupported="
  false" source="41" target="34" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="90" id="S1.3.11" toBeSupported="false"
  " source="42" target="41" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="91" id="C1.3.12" toBeSupported="
  false" source="44" target="34" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="93" id="C1.3.13" toBeSupported="
  false" source="47" target="34" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="94" id="S1.3.13" toBeSupported="false"
  " source="48" target="47" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="95" id="C1.4" toBeSupported="false"
  source="49" target="1" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="96" id="C1.4.14" toBeSupported="
  false" source="53" target="49" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="98" id="C1.4.15" toBeSupported="
  false" source="56" target="49" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="99" id="S1.4.15" toBeSupported="false"
  " source="57" target="56" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="100" id="C1.4.16" toBeSupported="
  false" source="58" target="49" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="101" id="S1.4.16" toBeSupported="
  false" source="59" target="58" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="102" id="C1.4.17" toBeSupported="
  false" source="61" target="49" />
<argumentElement xsi:type="ARM:AssertedInference" xmi:id="104" id="C1.4.18" toBeSupported="
  false" source="64" target="49" />
<argumentElement xsi:type="ARM:AssertedEvidence" xmi:id="105" id="S1.4.18" toBeSupported="
  false" source="65" target="64" />
</ARM:Argumentation>
```



THE ARGUMENT OUTLINE TEXTUAL REPRESENTATION OF THE PROCESS-BASED ARGUMENT

CLAIM C1: THE PROCESS MEETS THE SAFETY REQUIREMENTS.

CLAIM C1.1: THE ACTIVITY PROCESS_ARGUMENT_GENERATOR HAS BEEN CARRIED OUT.

CONTEXT C1.1: Standard: ARP4761 -- Performed @ 09/03/2016 18:21:21

STRATEGY RC1.1: Argument over roles

CLAIM C1.1.1: ROLE SAFETY ENGINEER IS CERTIFIED.
[Undeveloped]

STRATEGY WC1.1: Argument over work products

CLAIM C1.1.2: Process-Based SACM Argument IS AVAILABLE.
EVIDENCE S1.1.2: processBasedSafetySACMArgument.xmi

CLAIM C1.1.3: Textual Argument IS AVAILABLE.
EVIDENCE S1.1.3: processBasedSafetyArgument.txt

STRATEGY TC1.1: Argument over tools

CLAIM C1.1.4: TOOL PROCESS_ARGUMENT_GENERATOR -VERSION: 1 IS QUALIFIED
[Undeveloped]

STRATEGY GC1.1: Argument over guidance

CLAIM C1.1.5: GUIDANCE APPENDIX B3 HAS BEEN FOLLOWED.
EVIDENCE S1.1.5: Appendix B3

CLAIM C1.2: THE ACTIVITY ARGUMENTS COMPOSITION HAS BEEN CARRIED OUT.

CONTEXT C1.2: Standard: ARP4761 -- Performed @ 09/03/2016 18:21:21

STRATEGY RC1.2: Argument over roles

CLAIM C1.2.6: ROLE SAFETY ENGINEER IS CERTIFIED.
[Undeveloped]

STRATEGY WC1.2: Argument over work products

Appendix L: The Argument Outline Textual Representation of the Process-Based Argument

CLAIM C1.2.7: Safety Case Argument Fragment IS AVAILABLE.
EVIDENCE S1.2.7: CombinedSafetySACMArgument.xmi

STRATEGY TC1.2: Argument over tools

CLAIM C1.2.8: TOOL ARGUMENT_MERGER -VERSION: 1 IS QUALIFIED.
[Undeveloped]

STRATEGY GC1.2: Argument over guidance

CLAIM C1.2.9: GUIDANCE APPENDIX B3 HAS BEEN FOLLOWED.
EVIDENCE S1.2.9: Appendix B3

CLAIM C1.3: THE ACTIVITY FPTC_based Analysis HAS BEEN CARRIED OUT.
CONTEXT C1.3: Standard: ARP4761 -- Performed @ 09/03/2016 18:21:21

STRATEGY RC1.3: Argument over roles

CLAIM C1.3.10: ROLE SAFETY ENGINEER IS CERTIFIED.
[Undeveloped]

STRATEGY WC1.3: Argument over work products

CLAIM C1.3.11: FPTC results IS AVAILABLE.
EVIDENCE S1.3.11: bscu.flamm

STRATEGY TC1.3: Argument over tools

CLAIM C1.3.12: TOOL FPTC_based Analysis -VERSION: 1 IS QUALIFIED.
[Undeveloped]

STRATEGY GC1.3: Argument over guidance

CLAIM C1.3.13: GUIDANCE APPENDIX B3 HAS BEEN FOLLOWED.
EVIDENCE S1.3.13: Appendix B3

CLAIM C1.4: THE ACTIVITY PRODUCT_ARGUMENT_GENERATOR HAS BEEN CARRIED OUT.
CONTEXT C1.4: Standard: ARP4761 -- Performed @ 09/03/2016 18:21:21

STRATEGY RC1.4: Argument over roles

CLAIM C1.4.14: ROLE SAFETY ENGINEER IS CERTIFIED.
[Undeveloped]

STRATEGY WC1.4: Argument over work products

CLAIM C1.4.15: Product-Based SACM Argument IS AVAILABLE.
EVIDENCE S1.4.15: productBasedSafetySACMArgument.xmi

CLAIM C1.4.16: Textual Argument IS AVAILABLE.
EVIDENCE S1.4.16: productBasedSafetyArgument.txt

Appendix L: The Argument Outline Textual Representation of the Process-Based Argument

STRATEGY TC1.4: Argument over tools

CLAIM C1.4.17: TOOL PRODUCT_ARGUMENT_GENERATOR -VERSION: 3 IS
QUALIFIED.
[Undeveloped]

STRATEGY GC1.4: Argument over guidance

CLAIM C1.4.18: GUIDANCE APPENDIX B3 HAS BEEN FOLLOWED.
EVIDENCE S1.4.18: Appendix B3



THE XML PSSA PROCESS MODEL

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2015 rel. 4 (x64) (http://www.altova.com)-->
<Process xmlns="http://ncl.ac.uk/namespase" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" ID="ID1" xsi:schemaLocation="http://ncl.ac.uk/namespase Process-v1.2.xsd">
  <Description>Safety Process</Description>
  <Start_with>560c154ae4b0564156c5d5bb</Start_with>
  <Elements>
    <Activity ID="560c154ae4b0564156c5d5bb">
      <Name>FPTC_based Analysis</Name>
      <Description>fault propagation analysis task using FPTC analysis</Description>
      <Interactive>>false</Interactive>
      <No_of_input_ports>1</No_of_input_ports>
      <No_of_output_ports>1</No_of_output_ports>
      <Wait_for_input>>false</Wait_for_input>
      <Responsible_role>Safety Engineer</Responsible_role>
      <Standard>ARP4761</Standard>
      <Guidance>Appendix B3</Guidance>
      <Tool_qualification>>null</Tool_qualification>
      <Version>1</Version>
      <In_ports>
        <In_port>
          <From_activity>>null</From_activity>
          <Artefact ID="560c154ae4b0564156c5d5b9">
            <Filename>bscu.flamm</Filename>
            <Filetype>flamm</Filetype>
            <Description>the input model for FPTC analysis</Description>
            <Version>1</Version>
          </Artefact>
        </In_port>
      </In_ports>
      <Out_ports>
        <Out_port>
          <Next_activity>560c154ae4b0564156c5d5bb</Next_activity>
          <Artefact ID="560c154ae4b0564156c5d5ba">
            <Filename>bscu.flamm</Filename>
            <Filetype>flamm</Filetype>
            <Description>the result of the FPTC analysis task</Description>
            <Version>1</Version>
          </Artefact>
        </Out_port>
      </Out_ports>
    </Activity>

    <Activity ID="560c154ae4b0564156c5d5bb">
      <Name>Product-Based_Argument_Generation</Name>
      <Description>product based argument generation from FPTC results</Description>
      <Interactive>>false</Interactive>
      <No_of_input_ports>2</No_of_input_ports>
      <No_of_output_ports>2</No_of_output_ports>
      <Wait_for_input>>true</Wait_for_input>
      <Responsible_role>Safety Engineer</Responsible_role>
      <Standard>ARP4761</Standard>
      <Guidance>Appendix B3</Guidance>
      <Tool_qualification>>null</Tool_qualification>
      <Version>3</Version>
      <In_ports>
        <In_port>
```

Appendix M: The XML PSSA Process Model

```

        <From_activity>560c154ae4b0564156c5d5bb</From_activity>
        <Artefact ID="560c154ae4b0564156c5d5b9">
          <Filename>bscu.flamm</Filename>
          <Filetype>flamm</Filetype>
          <Description>the FPTC results</Description>
          <Version>1</Version>
        </Artefact>
      </In_port>
    </In_ports>
    <In_port>
      <From_activity>>null</From_activity>
      <Artefact ID="560c154ae4b0564156c5d5b9">
        <Filename>hazardous_events.xml</Filename>
        <Filetype>xml</Filetype>
        <Description>the hazards to look for</Description>
        <Version>1</Version>
      </Artefact>
    </In_port>
  </In_ports>
  <Out_ports>
    <Out_port>
      <Next_activity>560c154ae4b0564156c5d5bb</Next_activity>
      <Artefact ID="560c154ae4b0564156c5d5ba">
        <Filename>productBasedSafetySACMArgument.xmi</Filename>
        <Filetype>xmi</Filetype>
        <Description>the SACM representation of the product-based argument
      </Description>
        <Version>1</Version>
      </Artefact>
    </Out_port>
    <Out_port>
      <Next_activity>>null</Next_activity>
      <Artefact ID="560c154ae4b0564156c5d5ba">
        <Filename>productBasedSafetyArgument.txt</Filename>
        <Filetype>txt</Filetype>
        <Description>the textual representation of the product-based argument
      </Description>
        <Version>1</Version>
      </Artefact>
    </Out_port>
  </Out_ports>
</Activity>

<Activity ID="560c154ae4b0564156c5d5bb">
  <Name>Process-Based_Argument_Generation</Name>
  <Description>process based argument generation</Description>
  <Interactive>>false</Interactive>
  <No_of_input_ports>0</No_of_input_ports>
  <No_of_output_ports>2</No_of_output_ports>
  <Wait_for_input>>false</Wait_for_input>
  <Responsible_role>Safety Engineer</Responsible_role>
  <Standard>ARP4761</Standard>
  <Guidance>Appendix B3</Guidance>
  <Tool_qualification>>null</Tool_qualification>
  <Version>1</Version>
  <In_ports>
    <In_port>
      <From_activity>>null</From_activity>
      <Artefact ID="560c154ae4b0564156c5d5b9">
        <Filename>process_model.xml</Filename>
        <Filetype>xml</Filetype>
        <Description>the XML process model</Description>
        <Version>1</Version>
      </Artefact>
    </In_port>
  </In_ports>
  <Out_ports>
    <Out_port>
      <Next_activity>560c154ae4b0564156c5d5bb</Next_activity>
      <Artefact ID="560c154ae4b0564156c5d5ba">
        <Filename>processBasedSafetySACMArgument.xmi</Filename>
        <Filetype>xmi</Filetype>
        <Description>the SACM representation of the process-based argument
      </Description>
        <Version>1</Version>
    </Out_port>
  </Out_ports>
</Activity>

```

Appendix M: The XML PSSA Process Model

```
        </Artefact>
    </Out_port>
    <Out_port>
        <Next_activity>null</Next_activity>
        <Artefact ID="560c154ae4b0564156c5d5ba">
            <Filename>processBasedSafetyArgument.txt</Filename>
            <Filetype>txt</Filetype>
            <Description>the textual representation of the process-based argument
        </Description>
        <Version>1</Version>
        </Artefact>
    </Out_port>
</Out_ports>
</Activity>

<Activity ID="560c154ae4b0564156c5d5bb">
    <Name>Argument_Merger</Name>
    <Description>merge product and process based arguments</Description>
    <Interactive>false</Interactive>
    <No_of_input_ports>2</No_of_input_ports>
    <No_of_output_ports>1</No_of_output_ports>
    <Wait_for_input>true</Wait_for_input>
    <Responsible_role>Safety Engineer</Responsible_role>
    <Standard>ARP4761</Standard>
    <Guidance>Appendix B3</Guidance>
    <Tool_qualification>Qualified</Tool_qualification>
    <Version>1</Version>
    <In_ports>
        <In_port>
            <From_activity>560c154ae4b0564156c5d5bb</From_activity>
            <Artefact ID="560c154ae4b0564156c5d5b9">
                <Filename>productBasedSafetySACMArgument.xml</Filename>
                <Filetype>xmi</Filetype>
                <Description>the product based SACM argument</Description>
                <Version>1</Version>
            </Artefact>
        </In_port>
        <In_port>
            <From_activity>560c154ae4b0564156c5d5bb</From_activity>
            <Artefact ID="560c154ae4b0564156c5d5b9">
                <Filename>processBasedSafetySACMArgument.xml</Filename>
                <Filetype>xmi</Filetype>
                <Description>the process based argument</Description>
                <Version>1</Version>
            </Artefact>
        </In_port>
    </In_ports>
    <Out_ports>
        <Out_port>
            <Next_activity>null</Next_activity>
            <Artefact ID="560c154ae4b0564156c5d5ba">
                <Filename>CombinedSafetySACMArgument.xml</Filename>
                <Filetype>xmi</Filetype>
                <Description>the SACM representation of the merged argument
            </Description>
            <Version>1</Version>
            </Artefact>
        </Out_port>
    </Out_ports>
</Activity>
</Elements>
</Process>
```