

**Development Tools for context aware and Secure
Pervasive Computing in Embedded Systems
Middleware**

A THESIS
SUBMITTED TO THE SCHOOL OF
COMPUTING SCIENCE OF THE
UNIVERSITY OF NEWCASTLE UPON TYNE
IN PARTIAL FULFILMENT OF THE
REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Ran Zhao
October 2013

Abstract

The increasing number of devices that are invisibly embedded into our surrounding environment as well as the proliferation of wireless communication and sensing technologies are the basis for visions like ambient intelligence, ubiquitous and pervasive computing. The PErvasive Computing in Embedded Systems (PECES) project developed the technological basis to enable the global cooperation of embedded devices residing in different smart spaces in a context-dependent, secure and trustworthy manner. The PECES development tools aim to help the application developer to build applications using the PECES middleware and simulate the smart space dynamics such as device connections and context changes, etc. To ease the middleware development process, the development tools are implemented as Eclipse plugins and integrated into the Eclipse Integrated Development Environment (IDE). The development tools provide graphical user interface (GUI) to configure, model and test the PECES middleware based smart space applications. This thesis presents the design, implementation and devaluation of three groups of tools namely Configuration Tool (Peces Project, Peces Device Definition, Peces Ontology Instantiation, Peces Security Configuration, Peces Service Definition, Peces Role Specification Definition, Peces Hierarchical Role Specification Definition), Modelling Tool (Peces Event Editor, Peces Event Diagram) and Testing Tool which enable application developer to build, model and test the PECES middleware based smart space application using the novel concepts such as role assignment, context ontologies and security.

Acknowledgements

I would like to thank my parents they support me all the time. They also sponsor me all my cost when I living and study in a UK for several years.

I would like to thank my supervisor, Dr Neil Speirs, You have given me invaluable advice and guidance to help me finish my research.

I would like to thank my second supervisor, Dr Selva, you give me lots suggestions and help me finish my project.

I would like to thank all project partners who give the contribution in Modelling Tool development.

I would like to thank my close friends for helping me during the period of study.

Table of Contents

Abstract.....	i
Acknowledgements.....	ii
Chapter 1 Introduction	1
1.1 Background and Motivation	1
1.2 Challenges of supporting PECES Middleware.....	2
1.3 Objectives and Methodology	5
1.4 Requirements of the Development Tool	6
1.5 Contributions	9
1.6 Structure of Thesis.....	12
1.7 Publication History	12
Chapter 2: Background and Related work	14
2.1 Pervasive Middleware and Smart Space	14
2.1.1 State of Art	14
2.1.2 PECES Middleware	19
2.2 Context Ontology	27
2.2.1 State of Art	27
2.2.2 PECES Context Ontology Concept	31
2.2.3 Ontology Tools	35
2.3 Security Issue	36
2.3.1 State of Art	37
2.3.2 PECES Security Concept	39
2.4 Related Works	42
2.4.1 Pervasive Software.....	42
2.4.2 Other Development Tools.....	44
Chapter 3: Development Tool Design: Configuration Tool.....	47
3.1 Overview of Development Tools.....	47
3.2 Configuration Tool Introduction.....	51

3.2.1 Why we need configuration tool and what inside.....	51
3.2.2 Overview of Configuration Tool	52
3.2.3 Support for Role assignment architecture	55
3.3 Device Definition Tool.....	56
3.3.1 Device Definition Tool Prototype	56
3.3.2 Device Definition Tool Design	57
3.4 Ontology Definition Tool	59
3.4.1 Ontology Definition Tool Prototype	59
3.4.2 Ontology Definition Tool Design	61
3.5 Security Tool.....	63
3.5.1 Root Certificate Configuration	64
3.5.2 Intermediate Certificate Configuration	64
3.5.3 Client Certificate Configuration.....	65
3.5.4 Trusted Device Configuration	66
3.6 Service Definition Tool.....	67
3.7 Role Specification Definition Tool	69
3.8 Hierarchical Role Specification Tool	71
3.9 Cooperation with Modelling Tool.....	73
Chapter 4: Development Tool Design: Modelling Tools	74
4.1 Overview.....	74
4.1.1 Why we need modelling tool and what inside	74
4.1.2 Introduction of Modelling Tool	75
4.1.3 Support for Role assignment architecture	78
4.1.4 Cooperation with Configuration Tool	78
4.1.5 Discrete Event Dynamics Modelling	79
4.1.6 Cooperation with the Testing Tool.....	83
4.2 Event Editor	85
4.2.1 Overview Page	86

4.2.2 Context Page.....	86
4.2.3 Connection Editor Page.....	87
4.3 Event Diagram Editor.....	88
4.4 Summary	89
Chapter 5: Development Tool Design: Testing Tools.....	90
5.1 Introduction.....	90
5.2 Architecture	92
5.2.1 Execution Engine	93
5.2.2 Connection Engine	94
5.2.3 Event Engine	94
5.2.4 Instantiation Engine	95
5.2.5 Log Files.....	96
5.2.6 Analysis and Evaluation	97
5.3 Modules.....	98
5.3.1 Central Control Module.....	99
5.3.2 Middleware Modules.....	101
5.4 Testing Tool Implementation	103
5.4.1 Execute page	103
5.4.2 Testlog page.....	105
5.4.3 Visualisation page	106
5.5 Summary	113
Chapter 6: Use Case Study: build a real Application by using Development Tool	115
6.1 Introduction.....	115
6.2 Smart booth navigation.....	117
6.2.1 Scenario	117
6.2.2 Development by Development Tools.....	121
6.2.3 Application Prototype.....	123

6.2.3 Application Limitations.....	127
6.3 Smart Booth Monitoring.....	127
6.3.1 Scenario.....	128
6.3.2 Development by Development Tools.....	132
6.3.3 Application Prototype.....	134
6.3.3 Application Limitations.....	135
6.4 Smart Taxi Booking.....	136
6.4.1 Scenario.....	136
6.4.2 Development by Development Tools.....	138
6.4.3 Application Prototype.....	139
6.4.3 Application Limitations.....	142
6.5 Summary.....	142
Chapter 7 Evaluation.....	144
7.1 Evaluation Summary.....	144
7.2 Evaluation Methodological Approach.....	144
7.3 List of Associated Requirements Evaluation Results.....	147
7.4 Developer Feedback.....	151
7.2.1 Questionnaire Result.....	151
7.4.2 Questionnaire Result analysis.....	155
7.4.3 Productivity Plug-in.....	158
7.5 Summary.....	159
Chapter 8 Conclusions.....	160
8.1 Thesis Summary.....	160
8.2 List of Contributions and Benefits of Using PECES Development Tools	162
8.3 Future Work.....	166
8.4 Summary.....	167
References.....	168

Appendix 1 - project.xml.....	173
Appendix 2 - project.owl.....	175
Appendix 3 - GuideSYSTEMContext.pctx.....	176
Appendix 4 - events.xml.....	177
Appendix 5 - Questionary	180

Chapter 1 Introduction

1.1 Background and Motivation

The increasing number of devices that are invisibly embedded into our surrounding environment as well as the proliferation of wireless communication and sensing technologies are the basis for visions like ambient intelligence, ubiquitous and pervasive computing. The benefits of these visions and their undeniable impact on the economy and society have led to a number of research and development efforts.

These include various European projects such as EMMA [2], [3] that develop specialized middleware abstractions for different application areas such as automotive and traffic control systems or home automation. Middleware for pervasive environments primarily manages the stationary infrastructure in the environment. Usually, this infrastructure consists of stationary devices deployed within a predefined physical location such as meeting room or car parking. There are several other middleware systems that have been developed for this purpose such as Aura [19], Gaia [12], [13] and IROS [20]. Reliable service management framework is proposed in [21] by formally defining a message-oriented service application model and protocols that facilitate autonomous composition, failure detection and recovery of services.

These efforts have enabled smart spaces that integrate embedded devices in such a way that they interact with a user as a coherent system. However, they fall short of addressing the cooperation of devices across different environments. This results in isolated “islands of integration” with clearly defined boundaries such as the smart home or office. For many future applications, the integration of embedded systems from multiple smart spaces is a primary key to providing a truly seamless user experience. Nomadic users that move through different environments will need to access information provided by systems embedded in their surroundings as well as systems embedded in other smart spaces. The PECES project is committed to developing the technological basis to enable the global cooperation of embedded devices residing in different smart spaces in a context-dependent, secure, and trustworthy manner. The most innovative features of the PECES middleware are to enable the communication among

heterogeneous devices across the different smart spaces using dynamic addressing, security and context ontologies.

Pervasive computing application developers need development tools for rapid development and evaluation of novel smart space systems application. This is especially true for dealing with heterogeneous device environments with context based smart space formation as proposed in the PECES prototype applications. To the best of our knowledge, not many frameworks are available for effective simulation, emulation and testing of smart space system applications.

As describe above, a set of development tools is needed and can be used by the application developers to develop, test, and analyse their applications. The Development tools also provide an environment to simulate/emulate applications. These types of development tools are economical because developers can carry out experiments without the actual hardware and it is a feasible way to test scalability of any proposed applications.

The development tools focus on configuring devices, modelling smart spaces and context dynamics and testing the novel concepts provided by the PECES middleware. The tools provide support for application developers to build PECES middleware application and simulate and analyse the smart space behaviours with respect to the context changes and network changes. Instead of running PECES application on real devices, application developers are able to test the features of the PECES middleware in a development PC for any specific application. This provides the opportunity for the application developers to test and analyse their application in a controlled and repeatable environment which enable them to optimise certain parameters which may be necessary for the best performance of any smart space applications.

The PECES development tools provide a set of tools which are integrated into the Eclipse development environment (as Eclipse Plugins). This way, the usual development assistance provided by the Eclipse IDE can also be used for development support.

1.2 Challenges of supporting PECES Middleware

The lack of a generalized interaction mechanism between smart spaces restricts the availability of the remote functionality and access to the remote

services. Furthermore, due to the resulting lack of interaction, valuable context information may be lost when a user moves between different smart spaces. Yet, a user who may be interacting with several smart spaces in a day can be better supported by sharing context information. Thus, in order to realize the full potential of pervasive computing, smart spaces should support the interaction among devices without technical boundaries. Considering this every day user need, the PECES project develops a solution for enabling interaction with a smart space in the immediate vicinity of the user as well as on remote locations. Towards this end, the objective of the PECES project has been defined as the development a middleware that enables secure interaction among the devices in different smart spaces in context-dependent and trustworthy manner.

Besides supporting interactions among devices across the boundaries of smart spaces, the PECES middleware necessarily has to provide suitable solutions to several adjacent challenges of pervasive computing. Due to user mobility and the heterogeneity of devices that constitute a smart space, these challenges range from support for dynamic networks of embedded systems over interoperability to support for resource-poor and resource-rich devices. By providing a clear and uniform interface to application developers that is applicable to a broad range of devices, the middleware delivered by the PECES project greatly simplifies the task of the application developer in many application scenarios.

1) Context-aware middleware

The integration of devices across different smart spaces as targeted by the PECES project can support novel applications that combine services from different smart spaces. In order to discover and use these services in a distraction-free manner, PECES proposes the utilization of context information. As a consequence of this approach, there needs to be a common shared understanding of the information that is exchanged between different smart spaces. To ensure that this understanding exists and to ensure that it can be flexibly extended to new application scenarios, PECES relies on context ontologies to represent the shared information. The ontologies are an integral part of the dynamic addressing and grouping scheme and they ensure that the addressed or grouped devices are sound. Furthermore, the use of ontologies

also ensures that there is a standard way of extending the context modeled for the application prototypes.

2) Dynamic Addressing and Grouping

The PECES middleware support the integration of devices across different smart spaces. Providing services and access to remote resource has several advantages. First, it allows context sharing and, thus, reduces the required input from the users. Secondly, it reduces the latency needed for collecting the data and customizing the services. Third, and most importantly, it allows collaborative services and functionalities that are combined across different smart spaces.

To establish a suitable cooperation model (as described in section 2.1.3) among devices, to identify the devices in the surrounding and to remotely locate and utilize the services and resources, we need a global addressing scheme that is able to identify specific devices or resources across the boundaries of smart spaces. Traditionally the addressing schemes are designed according to the topology of the network. Thereby, a part of the address – usually some form of prefix – is used to identify the network of the device. However, in order to dynamically form a smart space and to integrate new devices in the environment, it may also be necessary to support processes, content, interfaces or resource migration from one device or smart space to another.

3) Security Issue

A key objective of the PECES middleware is to provide a cooperation layer that enables seamless interaction and coordination among devices in and across smart spaces in a secure manner. Intuitively, this requires adequate security mechanisms. To achieve this objective, the PECES middleware extends the basic middleware prototype to derive a secure middleware prototype. Towards this end, this specification introduced a basic trust model that is used as basis for the concepts and mechanisms of the middleware. These mechanisms enable the secure interaction of devices. To enable this, they span the management of cryptographic keys, the authentication of information – specifically context information and role assignments, the secure data- and service-centric communication as well as role-based access control.

1.3 Objectives and Methodology

PECES aims at providing a novel software abstraction that enables context-aware remote interaction among heterogeneous devices across different smart spaces in secure and trustworthy manner. Moreover, PECES also provides supplemental tools that support the novel abstraction and facilitate its effective use. Thus, PECES offers a complete solution that simplifies the modeling, configuration and testing of the application development. The main objective for providing the supplemental tools is to enable the programmers to easily and efficiently develop a wide range of pervasive applications. In addition to the developer, these tools can help everyday users as well. The main research and work areas that play significant roles in enabling cross smart space communication are the context ontology, dynamic addressing and grouping, and security. Naturally, the tools that will be developed during the course of the project will be oriented towards these three areas.

Most of other development tools only provide limited support for PECES developers as they have been designed for different goals and concepts. Although many projects have proposed development tools to support for pervasive computing environment application development, only little methodological support offered for context-awareness and security. The application developers also need support for development of highly dynamic and adaptive pervasive computing environments. PECES development tools will specifically target the novel concepts used in the proposed PECES prototypes applications such as context ontologies, dynamic addressing and grouping and security issues. The PECES development tools will empower application developers to make effective use of the concepts and mechanisms provided by the PECES middleware.

The PECES development tools will be the plug-ins of Eclipse integrated environment with a set of suitable development tools for the pervasive computing approach integrated with the novel concepts such as context ontologies and access control policies. The PECES project will develop novel development tools which are specifically provide support for PECES application developers.

1.4 Requirements of the Development Tool

This section provides a detailed list of the requirements on the development tools. The list also contains a rationale for the requirements and their priority as well as acceptance criteria that clarifies the context and scope of the corresponding requirement.

Requirement 1	
Description:	The development tools should support the programming language of the middleware.
Rationale:	The development tools may not be able to support all programming languages. A useful subset is required.
Acceptance Criteria:	The development tools can be used to facilitate application development in the language that has been used to implement the middleware.
Priority Rationale:	The consistency in using the same programming language (for both the middleware and applications) will help avoid compatibility issues.

Requirement 2	
Description:	The development tools should provide support for the devices of the prototype applications.
Rationale:	The development tools may not be able to provide support for all kinds of devices. Thus, the selection of a relevant subset is required.
Acceptance Criteria:	The development tools should support the platforms used for the prototype applications.
Priority Rationale:	This requirement identifies a constraint on the selection of the least set of devices. The development tools should support at least the devices that are used in prototype applications.

Requirement 3	
Description:	The development tools should support the specification of policies to limit the distribution of context information.
Rationale:	The description of a service that is stored in the registry may contain context information about devices and users. For some services, it may be desirable to limit the distribution of the information to a particular set of devices or users.
Acceptance Criteria:	The development tools allow the specification of limitations on the distribution of the context information that describes a service.
Priority Rationale:	A tool is needed to specify these policies and is part of the core functionality.

Requirement 4	
Description:	The development tools should support the configuration of encryption keys.
Rationale:	Some devices may not support direct interaction with the user. As a result, some keys might have to be configured at deployment time
Acceptance Criteria:	The development tools enable the developer to configure encryption keys for devices.
Priority Rationale:	We need a tool to configure an encryption key if a user unable to do directly. The tool is not needed if the key can be configured automatically or a user can configure the key inside the application.

Requirement 5	
Description:	The development tools should support the specification of static device context.
Rationale:	Some devices may not support direct interaction with the user. As a result, some of their context information may have to be configured statically.
Acceptance Criteria:	The development tools support the specification of static device context.
Priority Rationale:	The devices in smart spaces will need context information such as user profile. A tool should allow to statically configure the context information.

Requirement 6	
Description:	The development tools should be integrated into an existing IDE.
Rationale:	Switching between an IDE and the development tools by PECES can be distracting for the developer. Thus, the tools should be integrated into a suitable IDE that provides traditional programming support, for example.
Acceptance Criteria:	The development tools are integrated into an existing IDE.
Priority Rationale:	The tool will simplify the process of developing applications for the middleware but it should be possible to create applications with a simple text editor.

Requirement 7	
Description:	The development tools should support the testing of group specifications.
Rationale:	The outcomes of an abstract group specification language may not be easy to understand for novice programmers. A testing tool should help them to test the specification in different scenarios.
Acceptance Criteria:	The development tools empower the developer to define and execute a group specification.
Priority Rationale:	A testing tool would help developers to easily create and test group specification but it is not a necessary requirement to implement the core functionality of the middleware.

Requirement 8	
Description:	The development tools should support the modelling of a set of networked smart spaces.
Rationale:	In order to test the group specifications, the development tools must support the definition of a set of smart spaces.
Acceptance Criteria:	The development tools enable the modelling of a number of smart spaces.
Priority Rationale:	The modeling tool will be helping in providing input for group specification testing. But the tool does not affect the core functionality of the middleware.

Requirement 9	
Description:	The development tools should use the context ontology to simplify the user interface.
Rationale:	The specification of static device context for configuration and testing purposes may be time-consuming. This can be simplified by using the context ontology to define possible values.
Acceptance Criteria:	The knowledge encoded in the context ontology is used to simplify the user interfaces.
Priority Rationale:	The tool to use context ontology will simplify the user interface, but it does not have any impact on the middleware functionality.

Requirement 10	
Description:	The development tool shall support debugging functionalities.
Rationale:	Debugging functionalities reduce the time of the application developers and accelerate development process.
Acceptance Criteria:	The development tools provide basic debugging functionality.
Priority Rationale:	This will help application developers but may not support for all the devices used in the application.

Requirement 11	
Description:	The development tool should support the graphical user interfaces of various devices and their interaction.
Rationale:	A graphical user interface can simplify the development of applications using different smart spaces and devices.
Acceptance Criteria:	Important variables and functionalities are represented graphically.
Priority Rationale:	This will make development tools more user friendly but this is not a key innovation.

1.5 Contributions

The main contributions of this work are summarised as follows:

1) Configuration Tool

The configuration tool is responsible for coordinating initial instantiation and reconfiguration of a device or smart space. The PECES configuration tool

enables application developers to specify the context elements of devices and access control policies during device configuration. The tool also enables the specification of the appropriate key and key generators.

1) Device Definition Tool

This tool provides a graphical user interface (GUI) for application developers to specify the device. The Peces Device Definition Tool can be used to define BASE/PECES middleware communication plugins such as IP, Bluetooth, ZigBee and device functionalities and also device names.

2) Ontology Instantiation Tool

The Ontology Instantiation Tool provides GUI mechanism to define static context information relevant to the device and this information is used by the PECES middleware context components during the model execution.

3) Security Configuration Tool

The Security Configuration Tool integrates the OpenSSL toolkit to enable application developers to generate keys and certificates for smart space applications. The Security Configuration Tool provides an interface to gather necessary information for root certificate, intermediate certificate (trust chain) and client certificate.

4) Service Definition Tool

The Service Definition tool provides a simple interface to the developers that allows the automatic generation of all the code needed to instantiate and make use of a PECES-based service.

5) Role Specification Definition Tool

The Role Specification tool provides an interface where developers can define the different rules that the application use to dynamically form groups of collaborative devices.

6) Hierarchical Role Specification Tool

The PECES Hierarchical Role Specification tool provides an easy method to create all the code necessary to instantiate this kind of “composed” smart spaces.

2) Modelling Tool

The main task of the modelling tool is to support application developers in specifying the applications. The PECES modelling tool provides application developers to model the execution environment with context elements, resources and application properties. The modelling tool also allows the specification the properties of the communication mechanism between the individual devices such as requirements on encryption and the type of data exchanged between the devices.

7) Event Editor Tool

Event Editor is used to edit single event definitions. Type, Contributing Devices, Description and Duration (Delay) can be defined in the wizard and later altered on the Overview Page. The Event Editor is a multipage editor, on the second, Context Page the context of the corresponding device can be changed if the event’s type is Device Context Change. On the third page, connection can be defined if the event’s type is Connection.

8) Event Diagram Editor Tool

Event Diagram Editor is used to define the sequence of the events when the developer has defined the needed events.

3) Testing Tool

The PECES testing tool allows application developers to configure devices using the configuration tool and generate execution environments using the modelling tools. The testing tool then enables application developers to test if the application adapts to changes in the execution environment in the desired manner. For this purpose, the application developers need to provide context changes for different scenarios.

1.6 Structure of Thesis

This thesis consists of seven chapters, a bibliography and an appendix. The appendix lists the files sample using and generated by development tool.

The eight chapters start with this introductory Chapter 1. Next a brief literature work survey is presented in Chapter 2. In it the reader is given an overview of some of the background and related work in this area. The PECES concepts are also stated in this chapter.

The Chapter 3 we introduced the first set of tools, configuration tool. It concerns several tools which can be use to configuration the application. The goal of the configuration tool is to define the application device, set context value, generate security certificate and keys, set service and role specification. The follow chapter, Chapter 4, describe the modelling toll which is used to specify the environment, model the dynamic change of context and device and set the sequence of device. In Chapter 5, the testing tool is introduced. The aim of testing tool is to execute, test, validate and evaluate the application developed by configuration tool and modelling tool.

The last content chapter, Chapter 6, demonstrates how development tools help to develop a group of use case, trade show system. Trade show system has three sub application, Smart Booth Navigation, Smart Booth Monitoring and Taxi booking. This chapter shows the design, setting and how development tool work for them.

The evaluation chapter, Chapter 7, shows the evaluation design, process, result and the analysis of the result. The feedback from developers who attend test of development tools is also stated.

Finally in Chapter 8 we summarize the conclusions of the work presented in this thesis. We make recommendations for future work.

1.7 Publication History

Much of this thesis has been published in peer-reviewed publications and/or presented at conferences and journal. In this section we will list this publication history.

An early version of configuration tool was submitted to the International Conference on Wireless Information Networks and Systems (WINSYS2011). It was accepted and published in the proceedings.

The way of building application by early version of development tool has been submitted to Annual International Conference on Advances in Distributed and Parallel Computing (ADPC2011). It was accepted and published in the GSTF Journal on Computing (JoC - Print ISSN: 2010-2283, E-periodical: 2251-3043).

The extended version of Configuration Tool and Modelling Tool was submitted to International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2012). It was accepted and published in the proceedings.

The final version of tools which include all features in testing tool was submitted in Journal of Network (JNW). It got accepted and will be published in special issue in 2013.

Chapter 2: Background and Related work

The Tools described in this thesis involves several aspects have not been studied before. However this is some background and related work available. In this chapter we will provide reference for the background and outline the related research.

First, we talk about the research status of pervasive middleware and smart space and the features of PECES middleware in Section 2.1. In Section 2.2, we illustrate the development of Context ontology and show the PECES basic ontologies which may be used in Configuration tool. In Section 2.3, we discuss the security issue and some features about PECES security concept. We also show some pervasive system software and other development tool in Section 2.4.

2.1 Pervasive Middleware and Smart Space

2.1.1 State of Art

Pervasive computing middleware can be defined as a software layer between the operating system and the applications running in smart space. Thereby, pervasive computing middleware provides additional abstractions and services that support several applications and scenarios. To provide through support for the application developer, the development of pervasive computing middleware encompasses unique design considerations. Usually, a smart space consists of heterogeneous devices and platforms that might run a variety of operating systems. A middleware should hide the complexities of these underlying systems to provide a unified interface to the applications to facilitate portability and to lower the development effort. In addition to that, existing pervasive middleware typically exhibits different characteristics that are motivated by differences in their design goals. These characteristics include varying degrees of transparency, support for context management, adaptability or quality of service. In the following, we will discuss some of the most important existing middleware systems. Thereby, we classify the systems according to their cooperation models. There are two existing cooperation models for smart spaces are smart environments and smart peers.

- Smart Environment

Initially researcher in the pervasive computing area focused on the development of so-called smart environments. Smart environments define boundaries based on a physical location. Thus, they integrate devices in a spatially limited area such as a building or a room. Due to the static definition on the basis of location, the cooperation model of a smart environment usually relies on the permanent presence of a coordinating device to mediate the interaction of devices. Thus, this type of smart spaces provides us a location-centric and infrastructure-based solution. The services provided by these environments are location based. The smart environments are immobile in the sense that the group of the devices must remain in the close proximity e.g., in a smart conference room or garage, the embedded devices are attached to the physical locations. In some cases, however, the working group of the devices may have mobility as a group e.g., in smart car, GPRS and infotainment system remains installed in the car but the car itself moves.

- Smart Peers

Smart environments can provide services in a reliable manner. However, they have the potential limitation that they are infrastructure-based and require special effort for the initial setup. Furthermore, smart environments are not well suited for scenarios where devices come in contact with each other for a limited period of time or where ad hoc connectivity of the user devices is required. The smart spaces that explicitly target support for such scenarios can be classified as smart peers. Smart peers [4] are those smart spaces in which users of common interest or goals in a close proximity use their devices to form a smart space in a ad hoc manner. In this respect, the smart peers are more people centric. A group of people with same goals and interests can conveniently create smart peers using their mobile devices such as PPCs and laptops. Smart peers do not require special infrastructure support and are tied to the user instead of the location. In other words, this type of smart space is user-oriented and “moves” with the user. Since this approach is decentralized, the devices in the smart peers require a distributed coordination scheme to manage their services. This may introduce a considerable runtime overhead when compared with smart environments. Another issue with this kind of smart space is the

availability of the devices. Due to the ad hoc establishment of smart spaces the continuous availability of a single device and the presence of a certain set of necessary devices cannot be guaranteed which increases the complexities that application developers are facing when building applications for this cooperation model.

1) Middleware for Smart Environment

Middleware for smart environments primarily manages the stationary infrastructure in the environment. Usually, this infrastructure consists of stationary devices deployed within a predefined physical location such as meeting room or car parking. In addition, the middleware supports the dynamic integration of mobile devices carried by a user. Such devices may encompass mobile phones, PDAs and laptops. The resulting smart spaces enable the seamless interaction of the mobile devices with the stationary devices as long as they are within the same physical area. Three main representatives for these types of systems are Aura, Gaia and IROS which we briefly describe in the following.

Aura [19] is a middleware that focuses on providing services in non-intrusive manner. The design of the system is layer based. A layer observes a demand from above layers and self tune itself by anticipating the possible requests. For acquiring this proactive and self tuning behaviour, the Aura system requires knowledge about the environment as well as the knowledge about user intention. For example, suppose a user is watching a movie stream on network connection and suddenly the network speed slows down. In this case, there are a number of possible responses to the situation: 1) the user can be shown the movie on lower resolution, 2) the video can be paused to first perform buffering, or 3) the video can be stopped playing. The selection of the right choice mostly depends on not only the user context information such as her current task but also on her intent. Aura uses a task layer that works as a liaison between user and the rest of the layers. The main purpose of this layer is to anticipate user intent and to control the system accordingly. Aura provides high mobility for the users with persistent services across different Aura spaces. Aura is not designed to support specific scenario but is built to support different research

themes that include energy efficiency, user interface adaptability, task driven computing, resource amplification and wearable computing.

Another example of middleware for smart environments is Gaia [12][13]. Gaia is a system that provides services and infrastructure necessary to build general purpose pervasive applications. The application framework supports adaptation, mobility and dynamic binding. Overall, the Gaia kernel provides five basic services that are: event manager, presence service, context, space repository and context file system. The event manager manages the distribution of the events in the space and implements a decoupled communication model by means of information suppliers and information consumers. A presence service is responsible for detecting and maintaining the information about the presence of physical (e.g, people, devices) and digital (e.g., application and services) entities. Handling of the context information for the user support is another important aspect of smart spaces. A context service in kernel is responsible for maintaining context information about the environment. The space repository stores properties of the resources in the environment. Applications may use the space repository service to find suitable resource for the processes. The context file system uses the context information about the application task to guide the application process. Using these services, Gaia tries to abstract the heterogeneity of the devices in the smart environment and exploit the resources in the environment in a uniform manner.

iROS [20] is a middleware that allows multiple devices and applications to exchange information. The communication in iROS is supported through event heaps that allow the subscription and auto expiration of the events. The main purpose of the event heaps is to support dynamic coordination of the applications. Besides event heaps, iROS has two more sub-systems namely, data heap and iCrafter. Data heap allows moving data on application screens of various devices whereas iCrafter allows the collaborative control over devices and application in a meeting sessions. Overall the loose commutation mechanism in iROS allows the indirections using event heap that is advantageous in case where closely tied interdependencies might cause a crash. However, it may be ill-suited for applications that require the tight cooperation of several different systems.

2) Middleware for Smart Peers

As discussed earlier, smart peers are not location centric but rather follow a user centric approach. This is an immediate consequence of the fact that smart spaces can be created and migrated together with the devices of the user. Thereby, the migration of the smart spaces often takes place seamlessly.

One example of middleware that is based on the concept of smart peers is BASE [9]. BASE is a service-oriented middleware that supports the adaptation of communication protocols and technologies. It provides a uniform access to remote application services and device capabilities. It treats the devices capabilities same as application objects that offer services. The middleware is itself highly configurable and can be deployed on resource-constrained devices such as sensors as well as resource-rich computers. BASE is designed to support spontaneous communication between the devices and can automatically switch commutation stacks in cases where a problem with the current communication stack occurs.

Another example for middleware that supports the idea of smart peers is one.world [4]. The focus of the one.world project is to support the development of adaptable applications that support adaptation of the applications if the physical environment or computational environment changes. The main objectives of the one.world design are to support high mobility of the users in physical world, to support ad hoc connectivity with different devices and applications and to facilitate sharing of information. The design focuses on achieving these three objectives in a way that the overall system is less obtrusive and all these tasks are accomplish seamlessly. One.world is using a service-based architecture. It provides systems services that are implemented on top of foundation services. There are four foundation services in one.world that are virtual machine, tuples, asynchronous events and environments. One.world uses the Java virtual machine to ensure the portability of the application across heterogeneous devices. Tuples provide a data representation with optionally typed fields. The event notification system asynchronously notifies the application about their contextual change. The last foundation service is the environment that is responsible for running applications and keeping them isolated from other running applications.

One.world builds system services on top of these foundation services. Query engine, structured I/O, remote events and check pointing services are used for searching, storing data, communicating and fault protection respectively. Besides these services two most important services are the discovery and migration services. The discovery service is used to discover and connect with existing services. The discovery process can easily exploit the tuples for discovery as tuples are self describing data model. Migration services facilitate the movement of the environment to another location and thus the design, unlike smart environment is not tied to specific location.

2.1.2 PECES Middleware

1). BASE Middleware

The PECES project consortium built the targeted cooperation layer on top of BASE[9] middleware. This enables the project consortium to focus the development efforts on the novel and innovative features of the PECES middleware. BASE is freely available as open source under BSD license which facilitates the necessary modifications and extensions and enables the free reuse – even for commercial exploitation.

The overall architecture of BASE is divided into three layers. At the highest layer – the application layer, local and remote application objects and system services interact with each other. This layer relies on the functionality offered by the middleware core which is represented by the micro-broker layer. The micro-broker layer, in turn, uses the capabilities of the plug-in layer to discover remote devices and to communicate with them. Figure 2.1 shows the BASE three layer architecture and its main components.

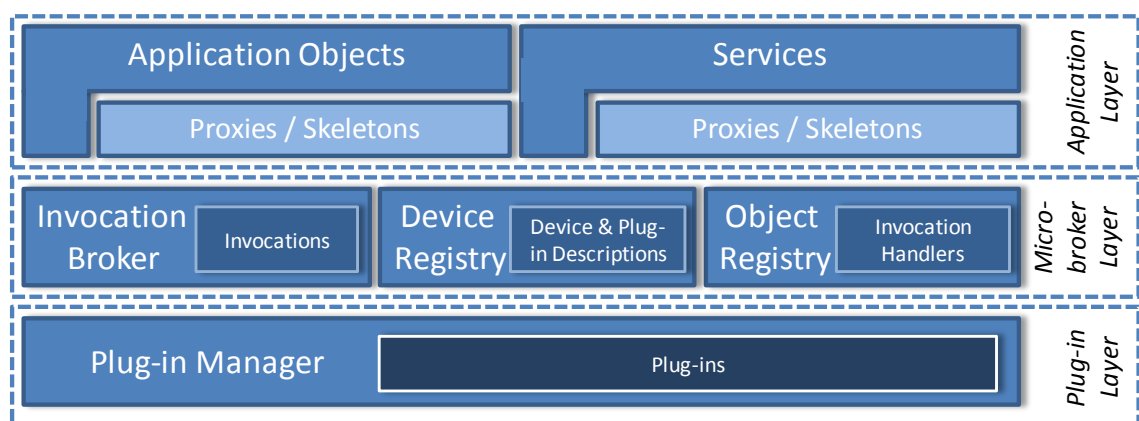


Figure 2.1 BASE Middleware Architecture

2). PECES Middleware Structure

The BASE middleware enables the communication between devices that are within communication range. Yet, in order to achieve the goal of providing cooperation layer that enables the seamless interaction within and across the boundaries of a single smart space, it is necessary to extend the basic concepts. The extension of the BASE middleware focused communication gateway concepts, addressing concepts and smart space concepts.

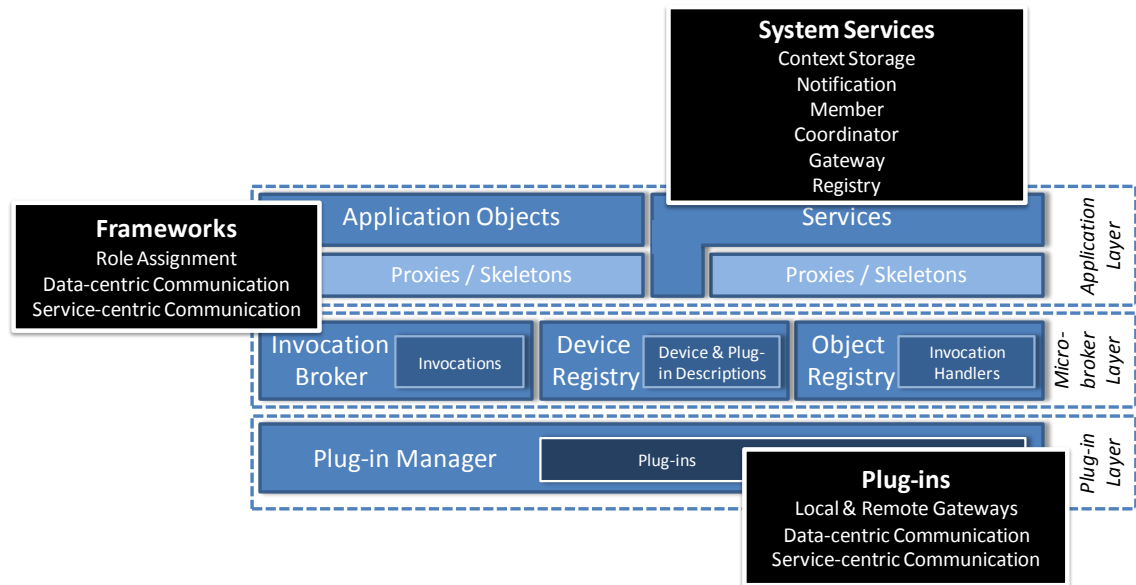


Figure 2.2 PECES Middleware Structure

At the plug-ins level, PECES provide follow extension communication capabilities of BASE middleware.

- Local & remote gateway plug-ins: The plug-ins provides support for routing and forwarding information among devices which in different smart space. For remote spaces, remote gateway plug-in requires information which is distributed by the registry.
- Data-centric communication plug-ins: This plug-in supports the transmission and reception of deta-centric messages which are exchanged between devices. A client and a server part is included in the data-centric plug-ins working with member and coordinator respectively.
- Service-centric communication plug-ins: This plug-in need to be installed in all devices whatever the device provide or use a service. It provides a request-response semantic for these devices.

At the higher level in middleware, system services provide several additional features to improve the middleware's functionality. The following services have been added:

- Context Storage: This service allow device to store context information that is available for role assignment process. Context information can be retrieved by coordinator in role assignment.
- Notification: The coordinator will call this service when there is any change such as a new role assigned or an existing role removed.
- Smart space: There are three different types of device in smart space are Coordinator, Member and Gateway. Each kind role has its own specific functionality. The coordinator service provides the ability to compute role assignments, the member service provides the ability to join and leave smart spaces and the gateway service provides the ability of connecting the space to the internet. Detail will be described in smart space concept section.
- Registry: Device, Space and Internet are three different level of registry. The registry is also implemented as a series of services. They provide the capability to access device level, smart space level or internet level service base on the scope of service. Device level registry is available on all devices. The smart space level registry work in smart spaces. And the internet level registry supplies more powerful devices which can access internet. Detail will be described in registry concept section.

The PECES also implement following frameworks to provide functionality to all applications:Role Assignment: The role assignment framework enables an application developer to formulate a role specification. The role specification can then be executed on the coordinator service which assigns the corresponding roles according to the specified constraints to the available devices.

- Data-centric Communication: The data-centric communication framework allows application developers to use role assignments to scope the distribution of messages.

- Service-centric Communication: The service-centric communication framework is used to support service invocations. This framework enable the usage of services among devices.

3). Communication Gateway

Due to the heterogeneity of devices and communication technologies, it is not safe to assume all future devices will be equipped with the same set of communication technologies. As an example consider that a sensor node might only be equipped with ZigBee (based on IEEE 802.15.4) but not with Bluetooth in order enable energy-efficient communication. Thus, in order to enable a Bluetooth device to communicate with such sensor nodes, it is necessary to use a device that is equipped with both technologies as a local gateway. Similarly, due to the associated costs and other factors, not all devices will have a direct connection to a global interconnection network like the Internet. In order to enable the communication between devices that are not directly connected to the Internet, it is necessary to enable some devices to act as remote gateways for others. Figure 2.3 shows a local gate way:

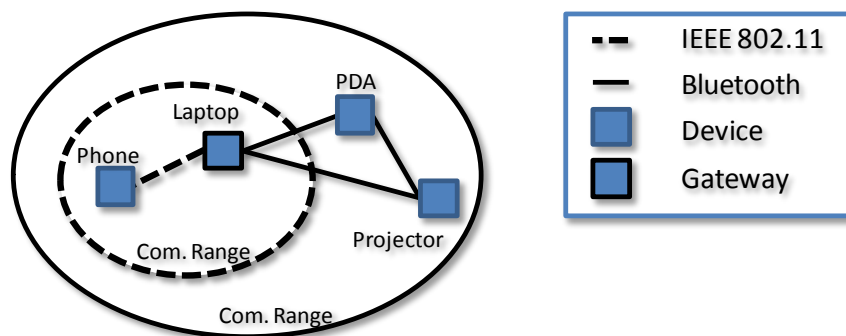


Figure 2.3 Local Gateway

PECES middleware support local gateways as well as remote gateways. The main difference between these two types of gateways is that the local gateway locally shares the required knowledge. In the remote case, the knowledge sharing should be restricted to a minimum in order to avoid the costly distribution of frequently changing information. The remote gateways need to be realized differently in that they require an external entity to distribute the information that is distributed by means of device discovery in the local case. This information is distributed by means of the registry that is specified in the

Figure 2.4 shows a remote game:

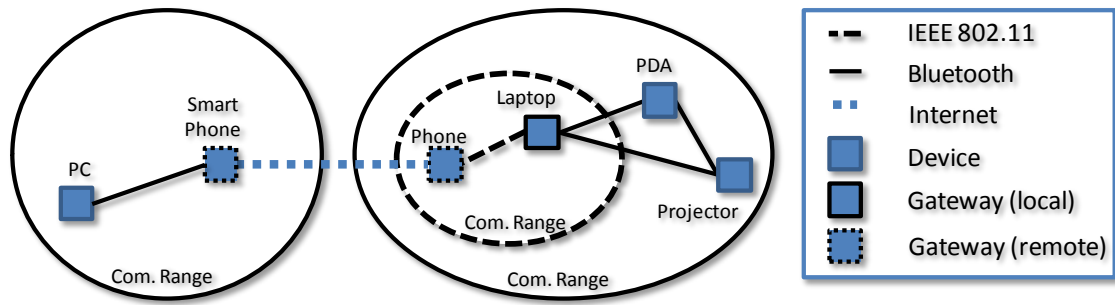


Figure 2.4 Remote Gateway

4). Generic Role Assignment Concept

Due to the continuous changes in context and due to the mobility of devices the underlying systems can be highly dynamic and the network topology can change frequently in the pervasive computing environments. So that it is vital to enable pervasive computing applications such as PECES prototype applications to adapt to the continuous changes in context and device availability. The responsibility for adaptation can be shifted between different entities. In cases where changes are infrequent, a user may manually configure and adapt the system. However, if changes are frequent, manual configuration and adaptation are clearly not a viable approach as they conflict with the goal of distraction free support for tasks. In order to mitigate this, the adaptation can be automated through the application. This approach relieves the user from performing manual adaptation but it complicates the development of applications and it may result in inefficiencies in cases where multiple applications implement and use similar adaptation mechanisms. As a result, the PECES middleware is aiming at automating the initial configuration and the continuous adaptation to changes in order to shield the user and the application developer from the accompanied complications.

In order to be suitable for a broad range of different systems and in order to minimize the utilization of resources that are required for automation, PECES middleware provides configuration and adaptation support by means of a uniform abstraction. To create a uniform abstraction that is suitable for a broad range of different configuration tasks, it is necessary to introduce a clear

separation between the result of a configuration, the computations that need to be done to produce it and the utilization of this result. This enables the reuse of the same basic mechanisms for different tasks. Generic role assignment provides such a uniform abstraction. More detailed information about the role assignment concepts can be found in [7].

A role can be assigned to any device as long as there are no further constraints that limit the assignment. To enable the automated computation of an assignment that reflects a particular goal of a configuration task, generic role assignment introduces rules. Rules define contextual constraints on the assignment of roles to devices. The simplest form of contextual constraint that is generally useful for all configuration tasks is a simple filter. An example of such a filter is to demand that all devices should be at a certain location. Another form of contextual constraint that is particularly relevant for PECES middleware are so called reference rules. Reference rules refer to a set of devices that has been assigned a particular role.

The set of rules together with their corresponding roles form a role specification. Given that the necessary contextual information can be captured by sensors or other types of information sources, one can use an algorithm to automatically assign roles to the devices whose context satisfies the constraints specified by their rules.

The architecture of the role assignment system consists of three main layers. These three main layers are context management, role assignment and service that use an assignment which as shown in Figure 2.5.

- Context Management: As describe above, system can automatic configuration and adaptation on the basis of context information by getting context information from sensors or other sources. The context management layer is responsible for gathering, inferring and fusing information that it receives from various sources. This resulting context information is then made accessible to the generic role assignment layer.
- Generic Role Assignment: The generic role assignment layer defines the concepts for the role specification. The role specification consists of roles and rules which are a set of conditions on which these roles are assigned to the devices. The role assignment layer uses context information to evaluate

these conditions. Besides assigning roles automatically to the devices, the role assignment system can also be used to assign the roles manually. Manual assignment of the roles provides more control to the user and is especially helpful in situations where manual configuration is desired.

- **Services:** Since roles are just tags, they are independent from their usage. In order to make use of a role assignment, we need to use them with other mechanisms. These mechanisms are specific to the configuration task. Thus, above the role assignment layer there are typically additional services that require assignments. To give some examples, for access control, the roles may be directly used as part of a role-based access control model. In the context of this specification, we will use them in order to address and group different sets of devices. This will enable the development of role-based communication mechanisms.

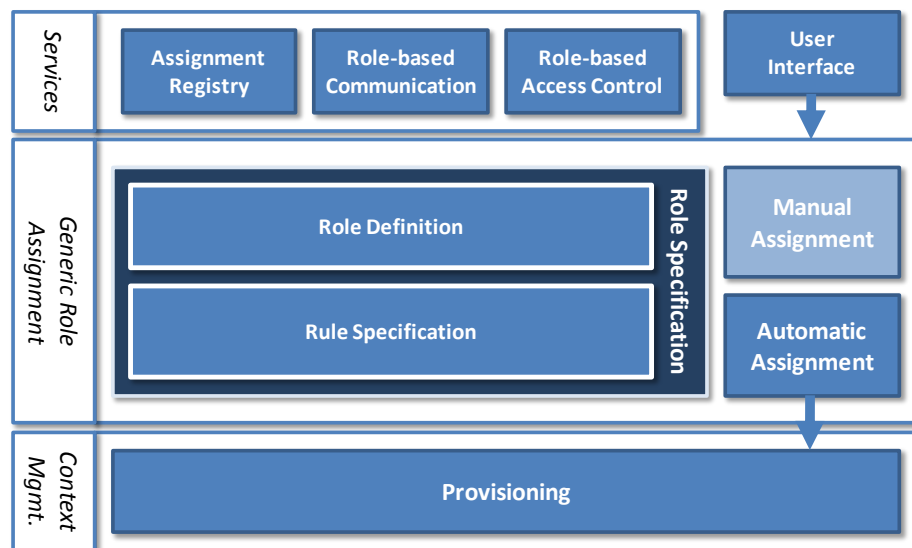


Figure 2.5: Role Assignment Architecture

5) Smart Space concept

A smart space can be defined as a group of networked devices that cooperate to support their users. The boundaries of a smart space are typically defined on the basis of a geographic location, e.g. a room or a building. However, such narrow definitions are not flexible enough to support the application prototypes in the PECES project. Obviously, these smart spaces cannot be defined on the basis of a single location. For example in applications based upon a car, the whole car, i.e. the smart space itself, is mobile.

In order to extend the definition, the addressing and grouping scheme can be used to support the formation of smart spaces based on arbitrary contextual properties. However, as explained earlier, the resulting definition is automatically restricted to devices that are residing in the same local network. This is a result of the fact that the formation process of basic groups is limited to a local network. Yet, for typical smart spaces local connectivity is guaranteed.

To support smart space formation, the PECES middleware introduces three additional components which are coordinator, member and gateway. These components can be easily motivated by looking at the anatomy of the smart spaces that are identified in the PECES Use-Case Specification:

Coordinator: A smart space consists of at least one coordinator device. This device is responsible for identifying members of the smart space based on role specification.

Member: In addition to coordinator device, a smart space may contain additional devices that are dynamically entering or leaving the local network. Depending on the context, a member device might either be integrated or not. Currently, a member device can only be integrated at most into one smart space at a time.

Gateway: Some devices that are part of a smart space may also be able to communicate with other devices through an Internet connection. Examples for such devices are smart phones or residential gateways as well as laptops that are equipped with a UMTS modem. In these scenarios, the PECES middleware gateway functionality provides connectivity for other devices in the smart space.

6) Registry Interface Concept

The PECES middleware provides a collaboration mechanism that enables communication between devices in and across different smart spaces in a context dependent manner. This requires a mechanism that allows devices to discover and access information about each other. The BASE middleware provides an internal service and device registries to maintain the access information of the locally available services. Since the services are accessed not only from inside but also outside the smart space, the PECES middleware provides a distributed registry mechanism that can further be extended for

remote group formation. The distributed registry can facilitate the information distribution of the assigned roles in the smart spaces coupled with necessary context information for forming an overarching environment. The PECES cooperation layer can thus use the distributed registry to lookup for devices based on the role and retrieve necessary plug-in information to access the devices. Hence, the distributed registry enables cooperation between heterogeneous devices by identifying the relevant devices that may provide services or can be used to form an overarching smart space on top of existing smart spaces.

The spontaneous appearance and disappearance of the devices in a typical smart space naturally requires a registry infrastructure where information about the services and roles are easily but securely accessible. The accessibility and security trade-off impose a natural scoping on the service availability. From a device perspective, the required services for an application may reside on the same device, or may be available on a remote device that may or may not be the part of the same smart space. This clearly outlines three different scopes for available services namely; “Device”, “Space” and “Internet”. More detailed information about the PECES registry concepts can be found in [8]. The registry Scopes and interaction shows in the figure 2.6:

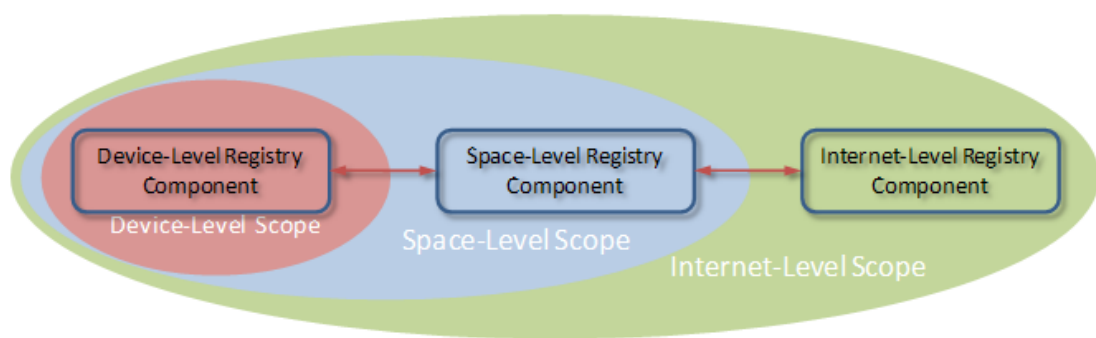


Figure 2.6 Registry Scopes and Interaction

2.2 Context Ontology

2.2.1 State of Art

Using context ontologies as an instrument for modeling contextual information provides an instrument to capture terms within domains and relationships among them in computer understandable formats. In order to leverage the

sharing of contextual knowledge and information reuse among PECES middleware instances contextual concepts, sub-concepts, relationships, properties and facts are represented in PECES ontology in a uniform way. This contextual knowledge can be later interpreted and evaluated by employing ontology reasoning which subsequently enables computers to determine the contextual capability, to compare contextual facts and to infer new and more complex context from primary measurements. There have been several related works on context ontology engineering in recent years and this section highlights the most relevant works to the PECES context ontology.

The PECES ontology aims to reuse as many as possible existing artifacts from off-the-shelf contextual ontologies, and thus we start with investigating which kind of concepts could be potentially reused in the PECES context ontology.

SOUPA [5] is an ontology which supports the largest amount of concepts because it builds upon the following existing relevant ontologies.

- FOAF[26]: This ontology captures an expression of personal information and relationships, and is a useful building block for creating information systems that support online communities [6]. Pervasive computing applications can use FOAF ontologies to express and reason about a person's contact profile and social connections to other people in their close vicinity.
- DAML-Time & the Entry Sub-ontology of Time [55][56]: The vocabularies of these ontologies are designed for expressing temporal concepts and properties common to formalization of time. Pervasive computing applications can use these ontologies to share a common representation of time and to reason about the temporal orders of different events.
- OpenCyc Spatial Ontologies [57] & RCC [58]: The OpenCyc spatial ontologies define a comprehensive set of vocabularies for symbolic representation of space. The ontology of RCC consists of vocabularies for expressing spatial relations for qualitative spatial reasoning. In pervasive computing applications, these ontologies can be exploited for describing and reasoning about location and location context [59].
- COBRA-ONT [59] & MoGATU BDI [60] Ontology: Both the COBRA-ONT and the MoGATU BDI ontology are aimed for supporting knowledge representation and ontology reasoning in pervasive computing environment.

While the design of COBRA-ONT focuses on modeling contexts in smart meeting rooms [61], the design of MoGATU BDI ontology focuses on modeling the belief, desire, and intention of human users and software agents [60].

- Rei Policy Ontology: The Rei policy language defines a set of deontic concepts (i.e., rights, prohibitions, obligations, and dispensations) for specifying and reasoning about security access control rules. In a pervasive computing environment, users can use this policy ontology to specify high-level rules for granting and revoking the access rights to and from different services [61]. Along with SOUPA, CONON [62] is an upper context ontology which defines 14 extensible core classes to model Person, Location, Activity and Computational Entities. Similarly, by focusing on Ambient Intelligence environments, the CoDAMoS [63] ontology focuses on the modeling of the user, environment, platforms and services aspects of its context information.

Another approach of using ontologies to model contextual information in applications and services is CAMidO [64]. CAMidDo uses ontologies to represent its meta-model in 3 tiers: middleware, context and applications. The first ontology is associated to the middleware level and concerns sensor descriptions. This ontology contains information about sensors that the middleware can interact with. This information is created and updated easily by the middleware maintenance agent to enable the middleware to interact with new sensors. Sensors can be used by all applications running on top of this middleware.

The second ontology is associated with the context level. It gathers information about context to which all context-aware applications, described using the CAMidO meta-model, are sensitive. The Context class belongs to this ontology. “Direct context” is captured directly from sensors, and “indirect context” is interpreted from other contexts.

The third ontology is associated with the application level and gathers information specific to the application allowing the designer to describe the following:

- All relevant contexts to which the application is sensitive by creating instances of the RelevantContext class.

- Context-aware components belonging to the application by creating instances of the Component class, and binding them with the described relevant context using the AwareOf property.
- Interpretation rules for indirect context description, by creating instances of the Policy&Rule class to describe interpretation methods and the HowDeduce property which enables the specification of indirect context that can be deduced using context information.
- Reactive adaptation of the application, when a relevant context is detected, by describing the ReactAdapt property which binds relevant context with the associated adaptation method described in the Policy&Rule class.
- Proactive adaptation of the application, by describing the ProactAdapt property which binds relevant context with the policy to be invoked (described as instance of the Policy&Rule class), and the component (belonging to the Component class) which invokes the service described in the CAService class.
- Context-aware services installed on top of the CAMidO middleware and context to which they are sensitive, by creating instances of the CAService class and binding each instance to the relevant context to which it is sensitive using the DependsOn property.

As in most of the proposed context ontologies, OWL-DL [41] seems to be a natural choice to model ontologies for its ensured decidability and as it is a W3C recommendation. However, RDF [65] or RDFS [66] are also candidates for modeling the simple relationships and taxonomies of contextual information. On the other hand, in some cases, expressiveness of OWL-DL is not enough to represent the contextual information. Furthermore, rule languages can be combined with OWL which has been proposed in SWRL [67].

However, the more expressiveness the ontological language supports, the higher demand of resource the hosting system should have to process.

Hence, available tools for fully processing ontological information can only be hosted in PCs or servers. So the main component of these tools that ontology-driven application employs is reasoner. The reasoner enables inferring implicit information defined by logic rules of ontological language. [68] has classified the

such support tools with respect to the underlying logic used ontological language. In principle we can differentiate between two branches of formal languages to use in ontologies: First-Order Logic (FOL,[69]) and Logic Programming (LP, [70]). FOL does adhere to the open world semantics and the non-unique name assumption, whereas the LP based languages usually do not. Description Logics (DL) is a subset of FOL. The strength of DL lies in subsumption reasoning and consistency checking. For classification and satisfiability checking there are mature reasoners available, while there is a lack of support for efficient instance retrieval.

2.2.2 PECES Context Ontology Concept

The PECES context ontologies **Error! Reference source not found.** are constituted from sub-ontologies such as Smart Space, Measurement, Device profile, User Profile and Event. Figure 2.7 shows the dependencies among them as well as external ontologies basing on which they extend concepts and properties. The external ontologies include wgs84_pos[25], OWL-S[24], OWL-Time[27] and FOAF[26]. wgs84_pos provides terms and property for representing spatial context information. OWL-S contains a set of ontologies for presenting services provided by devices/smart spaces. OWL-Time and FOAF ontologies provide terms and properties to present events and user’s profiles as described in the Use Case Specification.

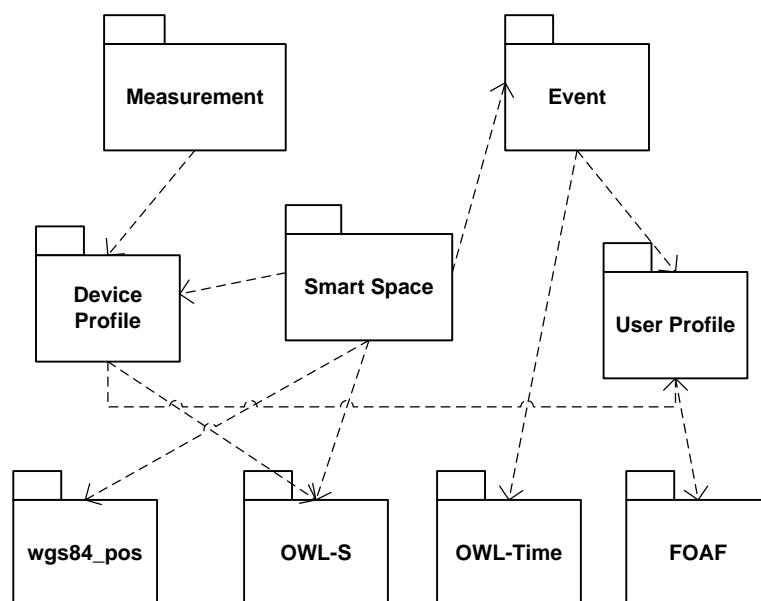


Figure 2.7 Dependencies of Context Ontology

The core ontology to describe the context information of smart space is Smart Space ontology. The *SmartSpace*, *Device*, *Location*, *Service* and *Context* are the basic concepts to model the context information of a smart space. All relationships among them are shown in Figure 2.8. The *SmartSpace* concept is used to extend to other sub kinds of smart spaces. *StationarySmartSpace* and *Non_StationarySmartSpace* are two major categories of smart space which are defined as subclass by *SmartSpace*. Smart space is defined as *StationarySmartSpace* when it having fixed location. Its location is referred to a location instance using *locatedAt* property. In the other hand, the smart space which does not have a fix location is defined as *Non_StationarySmartSpace*. The *Device* concept places the role as key abstraction for device profile ontology. *Context* concept is used to extend subclasses such as *LocationContext* or *SmartSpaceContext*. *Location* concept is used to representing the location information for *LocationContext* by using *relatedLocation* property and *StationarySmartSpace* by *locatedAt* property. The *Service* concept gets *ServiceGrounding*, *ServiceModel* and *ServiceProfile* from OWL-S ontology. When a smart space provides a service, *service:provides* property is used to connect *Service* instance to *SmartSpace* instance. And *service:providedBy* property is used to express a service is provided by a smart space.

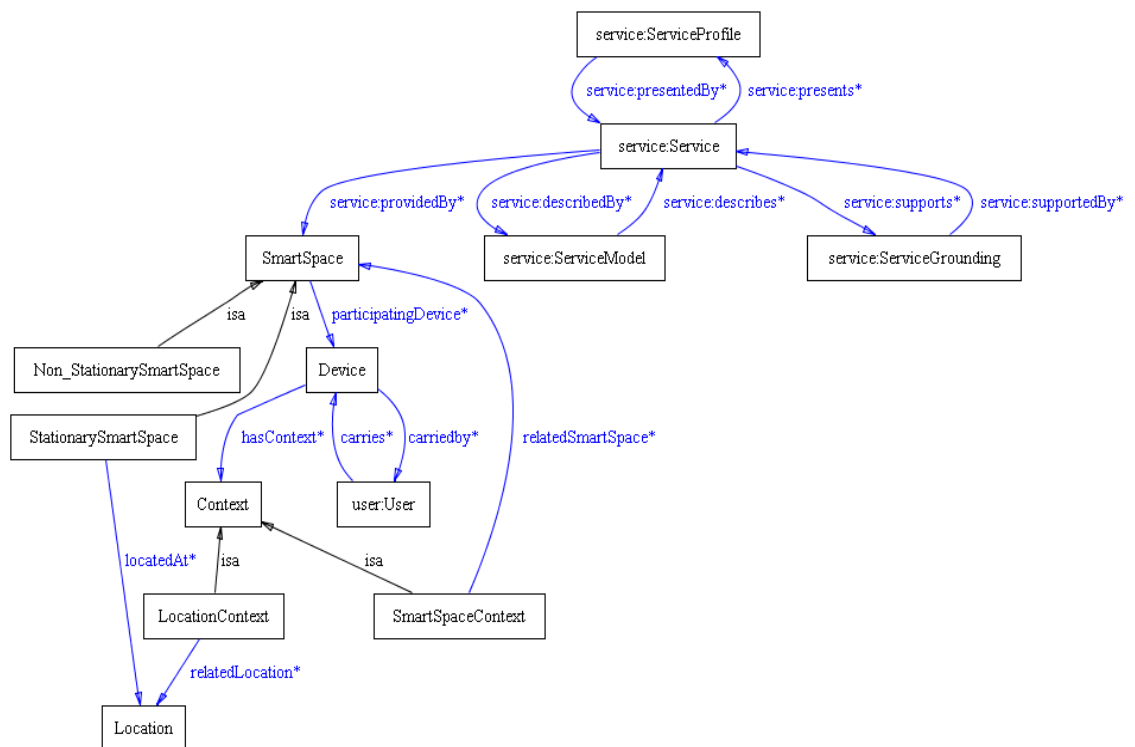


Figure 2.8 Core Contextual Concepts within a Smart Space

The Device profile ontology provides vocabularies to model specification of devices in smart spaces. *PECESEmbeddedDevice*, *Accessory* and *SensorDevice* are three kinds of devices. *PECESEmbeddedDevice* is the devices which has the PECES middleware installed. Property *service:provides* is used when the device provides a service. Three categories of embedded devices are defined in *PECESEmbeddedDevice* which are *Gateway*, *Coordinator* and *Member*. The *Accessory* property defined the accessories an embedded device has. Property *hasAccessory* is used to connect *Accessory* instance to *PECESEmbeddedDevice* instance. Input/output devices *Keyboard*, *Microphone*, *Touch Screen*, *Speaker* and *Screen* are defined as subconcept including in *Accessory* concept. The concept *SensorDevice* described the specification of sensor devices. The sub concept *MeasuringSensor* is used to represent a sensor which can measure a measurement. The concept *DeviceMobility* is a sub concept of the *Context* concept and which is used to form the mobility of device. The property *smartspace:hasContext* can link the Device instance to its mobility attribute. There are two types of mobility called *Non_Stationary* and *Stationary*. *Stationary* is used for device with fixed location. Vice versa, *Non_Stationary* is for representing devices may move. And *Attached* and *Portable* is defined as sub concepts for *Non_Stationary*. Figure 2.9 shows the relationship of Device Ontology.

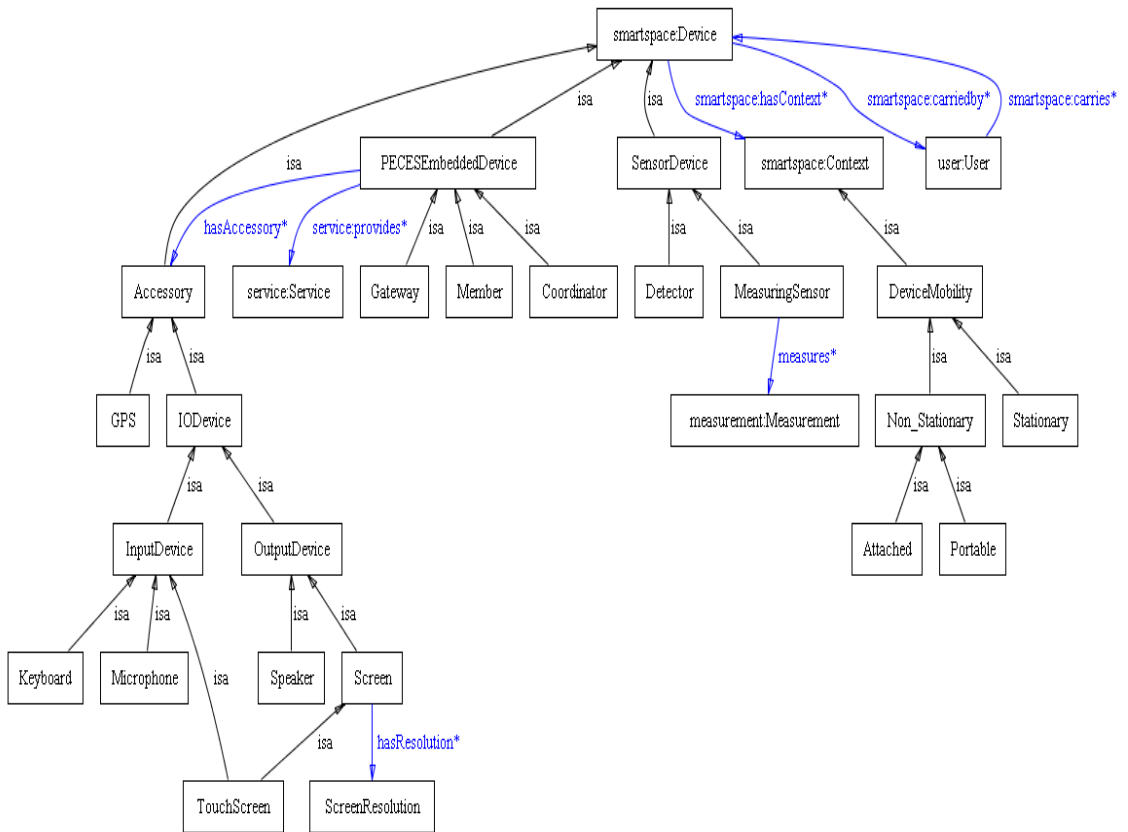


Figure 2.9: Device Profile Ontology

The Measurement ontology provides the categories of measurements which can be measured by sensor devices integrated in smart spaces. Figure 2.10 shows the graph of Measurement Ontology

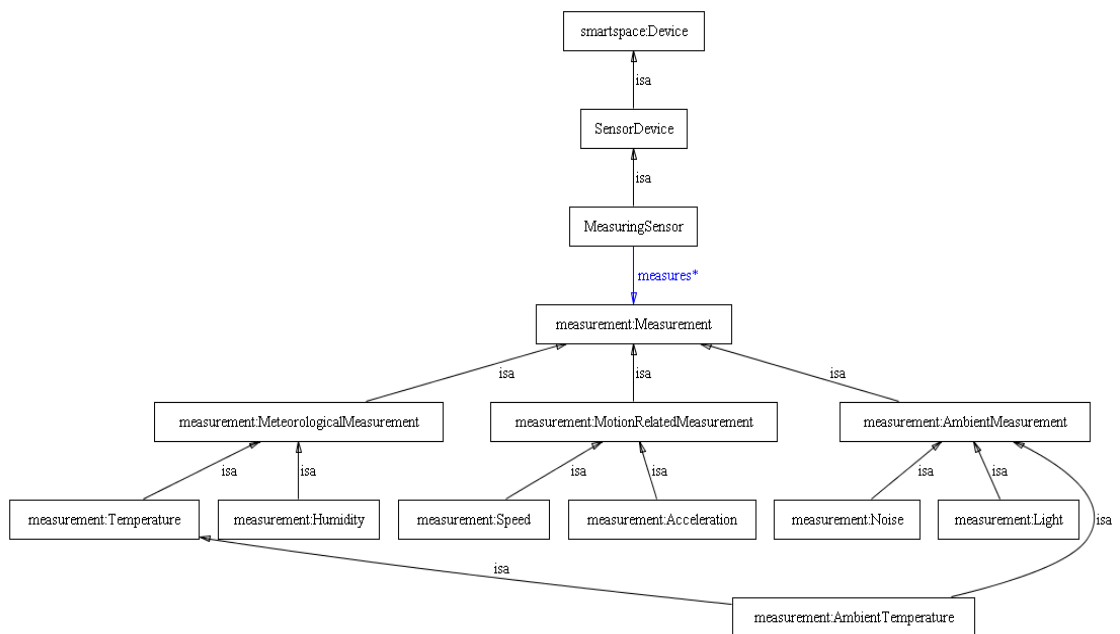


Figure 2.10 Measurement Ontology

The User profile and Event ontologies provide vocabularies to model the user's information and events in which user is involved. The User profile ontology extends FOAF ontology to identify the user's information. Thus all properties from concept *foaf:Person* are inherited by *User* concept, and any new properties can be added to *User* concept. The sub concept *user:UserProfile* and *user:Account* are linked to *User* concept as skeleton concept which can be added needed properties base on what kind of use case when developer want. The Event ontology employs the OWL-Time ontology to recognize event information. The *Event* concept is also defined as skeleton concepts which can be easy extend by further need. Figure 2.11 shows the graph of User profile and Event Ontology

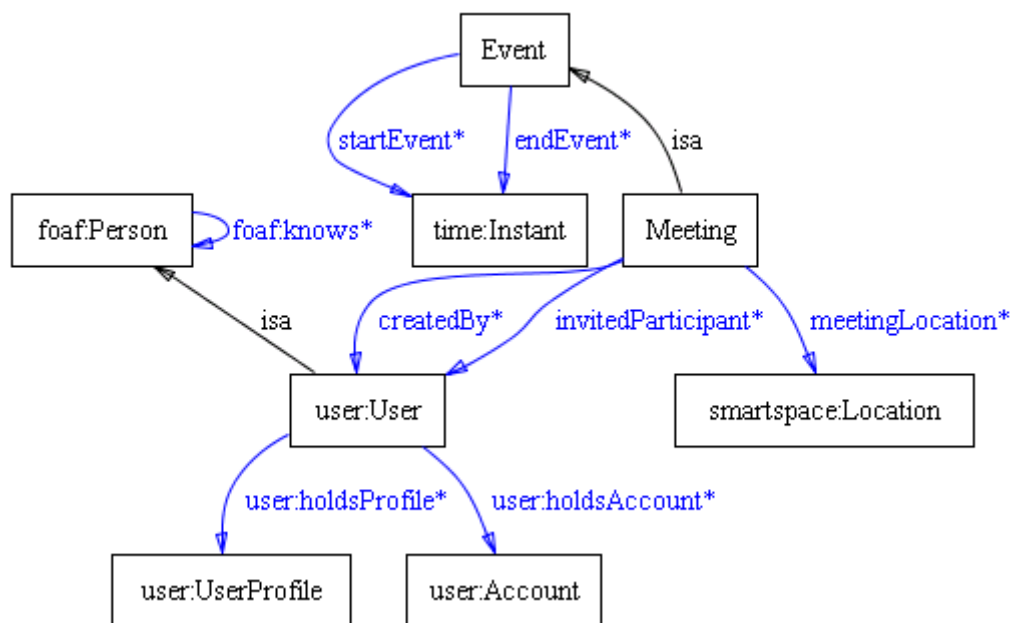


Figure 2.11 User profile and Event Ontology

2.2.3 Ontology Tools

a. Jena

Jena[45] is a Java framework building for semantic web applications originally developed by HP Lab. Jena support developers to handle several Ontology languages like RDF, OWL and SPARQL[28][31] by provides extensive java libraries. Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS [43] Ontologies, and a variety of storage strategies to

store RDF triples in memory or on disk. Jena is an open-source framework and has been extensively used in a wide variety of semantic web applications.

Jena was originally developed by researchers in HP Labs, starting in Bristol, UK, in 2000. Jena has always been an open-source project, and has been extensively used in a wide variety of semantic web applications and demonstrators. In 2009, HP decided to refocus development activity away from direct support of development of Jena, though remaining supportive of the project's aims. The project team successfully applied to have Jena adopted by the Apache Software Foundation in November 2010.

b. Protégé

A suite of tools used to construct domain models and knowledge-based applications with ontologies was provided by Protégé[22]. Protégé implements a rich set of knowledge-modelling structures and actions that support the creation, visualization, and manipulation of ontologies in many different representation formats. Protégé also provide a Java API for other developer to build their own tools and applications. Protégé support two different tools for modelling ontologies: Protégé-Frame editor and Protégé-OWL editor. Protégé-Frame editor provides a set of tools to build frame-based Ontology which consistent with OKBC (Open Knowledge Base Connectivity protocol)[44]. Protégé-OWL editor allows developers to build ontologies for Semantic Web. As its name as indicated by, Protégé-OWL editor is an extension of Protégé for supporting Ontology Web Language (OWL). This editor enables developers to load and save OWL and RDF ontologies, edit and visualize classes, properties, etc. In addition, Mayo Clinic provides some extension of Protégé that make Protégé can be embedded into eclipse as a plug-in.

2.3 Security Issue

Security and trust are essential parts of distributed system design. Although some security solutions for distributed systems can also be used for smart spaces, smart spaces usually require some additional considerations. Pervasive devices are deployed in the everyday surroundings and are accessible to everyone. This accessibility makes these devices vulnerable for hackers and prone to hostile attacks (such as denial of service). Furthermore, the devices in

a smart space have limited memory and processing power, and are often battery powered. These hardware limitations impose constraints on the technologies that can be used to provide secure services. Therefore, existing communication protocols that provide secure communication are not always suitable for smart spaces due to their resource requirements. Another crucial factor for security design is the distributed nature of the smart spaces. Heterogeneous devices and platforms add extra complexities. To achieve the goal of establishing secure and trustworthy communication of devices across the boundaries of a single smart space, it is therefore necessary to integrate appropriate security mechanisms and protocols.

2.3.1 State of Art

In many distributed systems, data is often transmitted through unreliable or unsecure networks. The way of dealing with insecure routes is to use cryptographic techniques. The two main classes of cryptography are symmetric key and asymmetric key cryptography. In symmetric cryptography, two entities share a same key. Entities in an authenticated network can use Kerberos [71] or Diffie-Hellmann key exchange [72] to establish keys. Different encryption algorithms can later be applied to ensure secrecy on the basis of the keys. AES, DES, Twofish, Serpent and Blowfish are some examples of widely used symmetric encryption algorithms. One drawback of symmetric keys is that they require an appropriate key management. However there are some approaches such as key exchange by means of physical location [73] or proximity [74] that simplify the key management. To utilize the context information of a node, Zhang et al [75] propose a location-based mechanism that assigns keys to static nodes based on their geographical location. In asymmetric cryptography which is also known as public key cryptography, the communicating entities use two different keys that are mathematically related. Although symmetric cryptography requires key management, it is usually considered to be more feasible for resources constraint devices. This is due to the fact that the asymmetric key cryptography usually requires more energy and memory to process the keys [76]. To mitigate this problem, the TESLA [77] suite, for example, introduces a mechanism that employs symmetric keys but later uses time to achieve asymmetry. Besides from key distribution, a second major problem of symmetric keys is that if a node is compromised, the whole network

becomes vulnerable. For this reason, some researchers shifted their focus to public key cryptography for resources constraint devices.

Besides standard security suites for traditional distributed systems such as IPSec [78] and Transport Layer Security (TLS) [79], there are several security suites that are specially designed for resource constrained devices. Examples are suites such as SPINS and TinySec. SPINS [80] proposes two protocols; SNEP and μ TESLA. SNEP (Secure Network Encryption Protocol) is used for authentication, and maintains confidentiality and freshness and μ TESLA performs broadcast authentication. In contrast to this, TinySec [81] provides link layer based security. However, TinySec is built for the TinyOS operating system and even though it can support small sensor nodes, it cannot be used for the communication between other types of devices such as PDAs and Industrial PCs. SensorWare [82] are other notable examples of security suite for sensor networks with similar characteristics.

For key management, Basagni et al proposed PebbleNets [83] that organize the network into clusters. Cluster heads from each cluster form a backbone. After that a Traffic Encryption Key (TEK) is generated by one cluster and forwarded along the network backbone. Cluster heads later forward the TEK to their cluster nodes. TinyKeyMan [84] is also another implementation on TinyOS for establishing pair-wise keys. The Localized Encryption and Authentication Protocol (LEAP) [85] is another key management protocol for small embedded devices that supports multiple symmetric key mechanisms and allows in-network processing. The rationale behind using multiple symmetric key is that the messages in a network can have different security requirements (depending on their content and context) and therefore they may require different encryption key mechanisms.

Since smart spaces are accessible by anyone and are usually deployed on insecure locations, a hostile attacker might pretend to be a legitimate user and can try to access resources as well as sensitive context data. For such cases Roman, Zhou and Lopez presented an intrusion detection system [86] where nodes monitor the communications in their neighborhood. Another problem usually faced by systems that are accessible to public is denial of service attacks. Since smart spaces are typically resource constrained, we must be

able to identify the attacked node as soon as possible while trying to keep rest of the environment functional. Against these denial of service attacks, [87][88][89] have proposed different solutions.

Access control is the ability of a system to manage the access to computer resources. A common technique is to maintain Access Control List (ACL). An ACL contains a mapping or association of permissions to resources. Overall, there are three basic access control techniques: Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC). In DAC, the access to the resources is controlled by the resources owner which is contrast to MAC where the access to the resources is controlled by a central system. In Role Based Access Control (RBAC) [90], users are assigned roles where each role contains a set of permission to access resources. Generalized Role Based Access Control (GRBAC) [91] is a context-oriented extension to the RBAC system where not only the users but also environments and resources are assigned roles. GRBAC make context-based decision to assign access privileges to the roles. Another extension of the Role Based Access Control is Role Templates [92]. In Role Templates, permission or privileges to access a resource depends upon the contents of the resource. For example, Role Templates can only allow a sales manager to query data relevant to the sales agents. TMAC [93] and OrBAC [94] focus on collaborative environments. Zhang and Parashar [95] extended the RBAC model and assign access control by combining user permission and the context information. Different languages have been proposed to define access control policies such as the Trust Policy Language [96], the Role Definition Language [97], and the FAM/CAM language [98]. The Generalized Access Control Language (GACL) [99] provides a RBAC based solution to control access control decisions based on the system load. GACL measure the load of the systems and only allows a program to execute if there is sufficient system capacity available.

2.3.2 PECES Security Concept

The PECES consortium extends the PECES middleware to derive a secure middleware. For this, PECES consortium introduced a basic trust model that is used as basis for the concepts and mechanisms of the middleware. These mechanisms enable the secure interaction of devices. To enable this, they span

the management of cryptographic keys, the authentication of information – specifically context information and role assignments, the secure data and service centric communication as well as role based access control. Although they do not introduce additional interaction features, together they span the whole set of security related requirements that have been identified in the PECES Requirements Specification [ref] and thus, they are sufficient to be applicable to a broad range of scenarios.

The PECES security mechanisms are modular and they introduce a certain degree of configurability that can be leveraged by application developers for optimization purposes. This enables them to define application specific trade-offs between security and application performance.

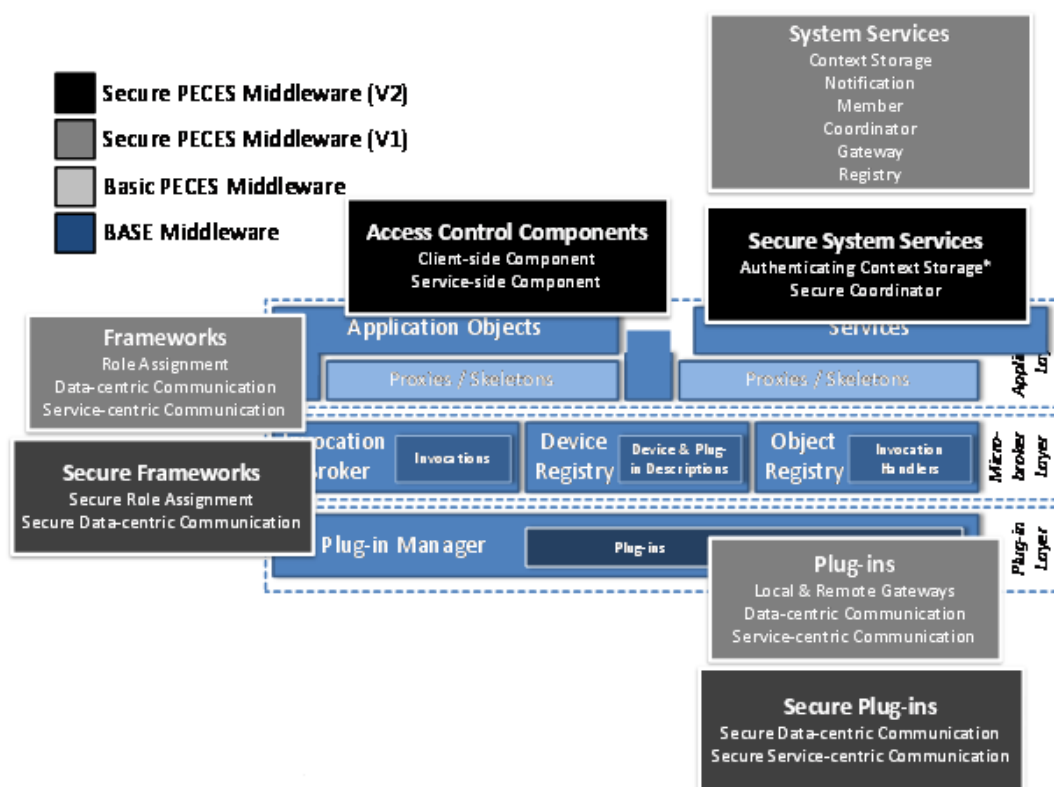


Figure 2.12 Secure PECES Middleware

At the plug-in level, the secure PECES middleware provides following extensions to improve the functionality of the basic PECES middleware which described in last sections:

- Secure Data-centric communication plug-in: This plug-in is an extension to the data-centric communication plug-in discussed in section 2.1.2. The secure version adds optional encryption capabilities, which can be

requested through the data-centric communication framework, to the basic plug-in. The framework also is extended for support this.

- Secure Service-centric communication plug-in: This plug-in is an extension to the service-centric communication plug-in discussed in section 2.1.2. Same as Secure Data-centric communication plug-in, this plug-in supply encryption capability for service-centric communication plug-in. The difference is the framework does not need to extend to support this plug-in. It only need the implementation provides some methods for simplifying the specification of security requirements before interaction between devices.

Secure system services provide following extra functionality of basic PECES system services extension:

- Key Storage and Management: The secure system service provide a new services named key storage and management service which supply the capability of store certificates for a particular trust level. This service not only stores the local keys but also temporary session keys. These keys will be used to authentication individual devices when key exchange.
- Authenticated Context Storage: The authenticated context storage service is an extension of the context storage part. When secure role assignments is needed, this service provide a function for devices which can give context information in an authentic way to the coordinator service. Thus coordinators can check the credibility of the context information they get and refuse the interaction from the devices with forged context information.
- Secure Smart Space (Coordinator): Secure Coordinator, an extension to coordinator, is used to secure role assignment and interaction of devices in a smart space. This gives the ability to coordinator to understand the security-relevant rules during role assignment. It also allows the coordinator to distribute the key when assigning a role.

For supporting the security concepts just talking above, the following features need to be added to basic frameworks:

- Secure Role Assignment Framework: As discussed above, coordinator system service is extended to support the security-relevant rules during

role assignment. These rules are implemented as part of the role assignment framework.

- **Secure Data-centric Communication Framework:** The secure data-centric communication framework added additional functionality to supply encrypts and decrypt messages with the group key distributed by coordinator before role assignment.

Secure middleware also includes the feature to support access control to services and information. The client-side component and service-side component are implemented:

- **Service-side Access Control Component:** The service-side access control component can validate the incoming requests and it can denies the unsecured requests.
- **Client-side Access Control Component:** The client-side access control component intercepts calls to service that are using service-side access control component. It can help the component to require the security privileges easily.

2.4 Related Works

2.4.1 Pervasive Software

a) DTT: A Distributed Trust Toolkit for Pervasive Systems

The Distributed Trust Toolkit (DTT) [11] proposed a framework for implementing and evaluating trust mechanisms in pervasive computing systems and introduced two new abstractions: trust groups and trust blocks. Trust groups allow associated application devices to share recorded trust data and trust computations. Trust blocks makes policy decisions based on data gathered by the computation component which implements network based trust protocols and allows the DTT to interoperate with legacy trust systems. The Distributed Trust Toolkit facilitates the extension and adaptation of trust mechanisms by abstracting trust mechanisms into interchangeable components. Furthermore, the DTT provides a set of tools and interfaces to ease implementation of trust mechanisms and facilitates their execution on a variety of platforms and networks.

b) Gaia: Enabling Active Spaces

The Active Space consists of the Gaia middleware OS [13] managing a distributed system composed of plasma displays, a video wall, audio system, touch screens, IR beacons, badge detectors, wireless and wired networks connecting several Windows 2000 and PDAs. The framework focuses on providing an application framework that leverages the functionality provided by the Gaia middleware OS to assist developers in the construction of Active Space application. The application framework defines an application model that accommodates the requirements of Active Spaces including dynamically changing the cardinality, location, input, output and processing devices used by an application. Then the application framework provides a mapping mechanism to define applications requirements and automatically mapping them to the resources present in a particular Active Space. Finally, the framework implements a flexible policy driven application management interface that allows customising applications to the dynamic behaviour of Active Spaces.

c) UBIWISE, A Ubiquitous Wireless Infrastructure Simulation Environment

UbiWise [15], a simulator for ubiquitous computing system was proposed in [14]. UbiWise concentrates on computation and communication devices situated within their physical environments. Multiple users can attach to the same server to create interactive ubiquitous computing scenarios. The devices are specified through a combination of a device-description file in XML and Java.

d) UbiREAL: Realistic Smartspace Simulator for Systematic Testing

UbiREAL [16] simulator was proposed for realistic smart space systematic testing. UbiREAL facilitates reliable and inexpensive development of ubiquitous applications where application software controls a lot of information appliances based on the state of external environment, user's contexts information. The simulator realistically reproduces behaviour of application software on virtual devices in a virtual 3D space. Interestingly, it provides mechanisms to simulate virtual devices with real devices.

e) A Middleware-based Application Framework for Active Space Applications

A Middleware based application framework for Active Space applications [12] was proposed in M. Roman's paper. The Active Space consists of the Gaia middleware OS managing a distributed system composed of plasma displays, a video wall, audio system, touch screens, IR beacons, badge detectors, wireless and wired networks connecting several Windows PCs and PDAs. The framework focuses on providing an application framework that leverages the functionality provided by the Gaia middleware OS to assist developers in the construction of Active Space application. The application framework defines an application model that accommodates the requirements of Active Spaces including dynamically changing the cardinality, location, input, output and processing devices used by an application. Then the application framework provides a mapping mechanism to define applications requirements and automatically mapping them to the resources present in a particular Active Space. Finally, the framework implements a flexible policy driven application management interface that allows customising applications to the dynamic behaviour of Active Spaces.

2.4.2 Other Development Tools

A development environment is a type of computer software that assists programmers to develop, build, deploy and analyse applications. The development environment normally consists of a source code editor, a compiler or interpreter, build-automation tools, and usually a debugger. Typically a development environment is devoted to a specific programming language, as in the Visual Basic, C++ or Java, although some multiple-language development environments are in use, such as Microsoft Visual Studio or NetBeans, In recent years, there has been emergence and popularization of Open Source development environment such as Eclipse and NetBeans.

Eclipse [10] is an open source development environment provides a robust, commercial quality platform for development and support software engineering. Generally Eclipse is frame-work for plug-ins. This plug-in architecture provides flexible and scalable tool integration with features to customize or extend a component to the developer's needs. The Eclipse Graphical Modelling Framework (GMF) and Graphical Editing Framework (GEF) provide a generative component and runtime infrastructure to build graphical modelling

editors. The Eclipse Platform reduces the cost of tool integration by providing a large number of services, APIs, and frameworks that enable effective and scalable tool integration.

The Eclipse Platform provides a focal point for integrating and configuring tools in a manner that best fits the end user's development process. Eclipse Workbench provides a central integration point for project control and an integration mechanism for resource-specific tools. This approach allows a user to build applications using a heterogeneous set of tools while at the same time providing a common view of the complete application across all components [5]. Eclipse UI integration allows a tool to participate with other tools as if they were designed as a single application. Tool integration is done by specifying at runtime the tools with which an application wants to integrate. It is believed that Eclipse is a cost-effective, productive development environment to integrate new tools for new project requirements.

There are several development tool have been proposed to support pervasive computing middleware application developers. CASA (Contract-based Adaptive Software Architecture) [100] middleware provides a framework for enabling the development and operation of pervasive applications. CASA Runtime System (CRS) is responsible for monitoring the execution environment and adaptation of the affected applications. The monitoring contextual information includes acquiring data, structuring the acquired data based on an application domain specific ontology and deducting the final knowledge. The contract-based adaptation policy used in CASA framework facilitates changes in the adaptation policy at runtime.

The Component Synthesis using Model Integrated Computing (CoSMIC) [101] is the Model Driven Architecture (MDA) based tools targeted primarily for distributed real-time embedded applications. The CoSMIC tools consist of an integrated collection of modelling, analysis and synthesis tools that address key lifecycle challenges of middleware and applications. The tools initially targeted CIAO [102] which is s QoS enabled CORBA component model middleware and QuO [103] framework which is an adaptive middleware for distributed real-time embedded systems. The CoSMIC tools provide a modelling and analysing

framework for key QoS properties of CIAO and QuO in the static and dynamic execution environments.

Music project [104] proposed development tools for adaptive applications in pervasive computing environment which support context changes and maintains a high level of usefulness across context changes. The project uses a model-driven approach for development of applications and services that utilize context ontologies. Model-based development requires modelling and transformation tools in order to automate the development process. An UML modelling tool, an ontology editor for services ontologies and transformation tools are used by the middleware to perform adaptations. Madam project [105] has developed modeling language extensions and tools enabling application designers to specify adaptation capabilities at design time. It also follows the model-driven approach and the source code is automatically generated by model transformation to pass the adaptation capabilities, context dependencies and application properties to the MADAM Middleware.

In the PLASTIC [106] project, the development tools are based on UML, JDK and Eclipse. It includes a model based testing tool that automatically derives and executes invocation sequences on a service and given response confirmation to a service state machine which can be modelled using commercial UML editor. It also includes a synthetic-workload generator for managing the deployment and execution environment. The PLASTIC development tools are implemented as Eclipse plug-ins and each tool has been developed in a modular way that would allow an application developer to use the tool outside Eclipse.

However, development tools described above only provide limited support for PECES developers as they have been designed for different goals and concepts. Although many projects have proposed development tools to support for pervasive computing environment application development, only little methodological support offered for context-awareness and security. The application developers also need support for development of highly dynamic and adaptive pervasive computing environments.

Chapter 3: Development Tool Design: Configuration Tool

In this chapter we describe the Configuration Tool which is used to start the application development. First of all, we give overview of Development Tool in Section 3.1. The Configuration Tool contains six tools. Section 3.2 talks about the architecture of Configuration Tool briefly. Device Definition tool is the first tool to setting the application device's attribute which will be described in Section 3.3. The Ontology Instantiation Tool, addressed in Section 3.4, is used to setting the static context properties. For security purpose, Security Tool can be used to generate the necessary certificate and keys, which is discussed in the Section 3.5. In Section 3.6, we show Service Definition Tool for helping developer use a PECES-based service. When developer needs to set the group rule set for smart spaces, Role Specification Tool will be used. It is described in Section 3.7. The last tool of Configuration Tool will be addressed in Section 3.8. It is Hierarchical Role Specification Definition Tool which possible to define smart space hierarchically. The Role Specification Tool and Hierarchical Role Specification Definition Tool has contributions from Alberto who from ETRA, Spain. We end with a brief summary of this chapter in Section 3.9.

3.1 Overview of Development Tools

Providing people with useful services, making embedded devices cooperate based on their context, is one of the most important challenges in smart space environments. Since applications running on smart space concepts have huge potential and convenience in our daily life, these applications must be developed carefully and tested before deploying for real world environments. However, it is very expensive to test them thoroughly in real world environments, since experiment setup have to assemble a Test bed using various types of sensors and embedded devices and generate a huge number of contexts for tests where each context consists of user locations, behavior, time, etc. The state of the art tools discussed in previous section can only be used to evaluate little aspects of the smart space applications in the PECES project. In order to cope with the applications proposed in the project and to test the novel features introduced by the PECES middleware, we provide a different set of tools for application developers.

PECES development tools focus on configuring devices, modelling smart spaces and context dynamics and testing the role specification concepts for the three proposed prototype applications. The tools are used for simulating and analyzing the smart space behaviours and context changes of the different application scenarios. The tools use the discrete event simulation concept which is used in several network simulations. Instead of running PECES application on real devices, application developer is able to test the features of the PECES middleware in a development PC for any specific application. This provides the opportunity for the application developers to test and analyse their application in a controlled and repeatable environment which provide information to optimise certain parameters which may be necessary for the best performance of any smart space applications.

The PECES development tools provide graphical user interface (GUI) features where application developers can configure and model any smart space applications by using a drag and drop method and edit information by clicking on the devices. The tools also provide GUI based test environment where application developers can start, stop and suspend the application and evaluate the application results. PECES development tools use the GUI device/model description information to generate XML description files (JDOM plug-in can be used in Java) which are used by the PECES middleware components during execution. The tools also provide mechanism for network dynamics, context changes which help application developer to understand the behaviour of the PECES middleware for any given application in different scenarios.

Providing a realistic simulator/emulator environment would be very useful for the application developers, but developing a complete realistic simulator/emulator is a large effort. It is noteworthy that PECES tools target novel features such as role assignment, context grouping and security mechanism introduced by the PECES middleware which is not impacted by wireless realistic simulations.

Due to the heterogeneity of devices that the PECES middleware addresses, it is not feasible to provide development tools that deal with low level details (for instance, emulation of specific hardware modules). As a result, the development tools only focus on testing the novel features of the middleware which the application developers will be interested when they are building application with

PECES middleware components. The tools do not concentrate on testing the prototype applications behaviour on the specific hardware platforms.

The PECES project provides a set of tools which are integrated into the Eclipse development environment. This way, the usual development assistance provided by the Eclipse IDE is enhanced with the new PECES focused development support. The PECES development environment have three tools namely Configuration Tool, Modelling Tool and Testing Tool to support testing the pervasive computing novel concepts introduced by the PECES middleware such as dynamic grouping, context ontologies and access control policies.

➤ ***Configuration Tool***

The Configuration Tool provides graphical user interface to configure the virtual devices which is used for testing a specific application. The Configuration Tool enables application developers to specify the device properties and initial context information during the device configuration. Keys and certificates are used by the security components of the PECES middleware (asymmetric and symmetric cryptography) are configured by the Configuration Tool as well. Device description information and context information can be used to model smart space defined by the role specification. Service running by devices also can be generated by Configuration Tool.

➤ ***Modelling Tool***

The Modelling Tool provides application developers to specifying the application scenarios which include environment, event and dynamics of the model. Role specification, connection dynamics and context changes are handled by the Modelling Tool. Smart space definitions and PECES global topology also has to be deployed by the Modelling Tool..

➤ ***Testing Tool***

The Testing Tool allows application developers to execute the application scenarios which are already defined by the Configuration Tool and the Modelling Tool. The Testing Tool also provides a mechanism to evaluate the test results. The Modelling Tool provides a ready to run test application to the Testing Tool which is responsible for the execution of the defined model and pass relevant data to the middleware component. The Testing

Tool allows application developer to start, stop and suspend the test application and provide mechanism to evaluate the application results.

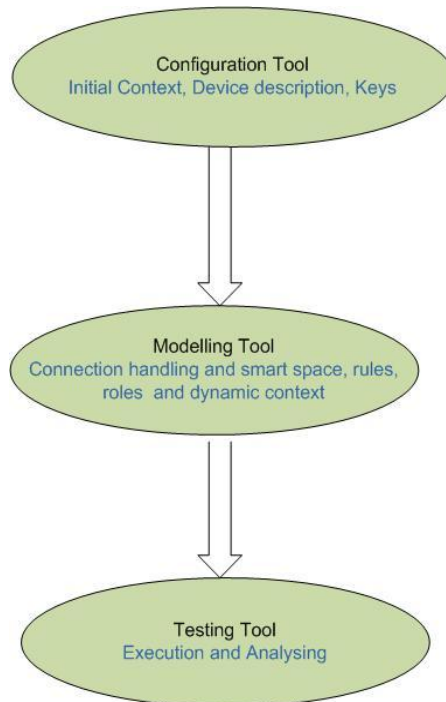


Figure 3.1: PECES Development Tools Interaction

The three tools discussed here are implemented in Java and provide configuration, modelling and testing environments to application developers. The tools interact with each other as shown in Figure 3.1. Initially, application developer will start with the Configuration Tool by defining devices, deploying keys, certificates, initial/static context information, role specification information and service. The output of the Configuration Tool will be used by the Modelling Tool which will add details about network dynamics and context changes which will be used by the middleware to form the smart space. Output of the Modelling Tool will be a test model which will be used by the Testing Tool to execute the define application scenario. The Testing Tool will execute the applications and provide evaluation mechanism for the test results.

Using PECES development tools, application developers will be able to model smart spaces and integration of the smart spaces. A smart space can be formed as a group of networked devices that cooperate to support any specific task defined by the role specification. To model a smart space in the development environment, application developers will have to configure several

devices using the Configuration Tool. A model smart space should consist of at least one coordinator device and some other devices such as gateway and member devices. In addition to these initial devices, a smart space may integrate additional devices which effectively form a different network topology. However, a device will be integrated only into one smart space at a time. Also two smart spaces (Smartspace 1 and Smartspace 2 in Figure 3.2) can merge as another smart space (Smartspace 4) but the new smart space will only have one coordinator. All these information have to be defined by the application developer in role specification phase and the Modelling Tool provides features for this. The application developers is able to visualise different smart spaces provided by the GUI in the Modelling Tool as illustrated in the Figure 3.2 below. If the application developers are not modelling the smart space properly and configure the devices incorrectly, the Modelling Tool and the Testing Tool provide some error messages to the developers which help them to re-configure and re-model their devices and application scenarios.

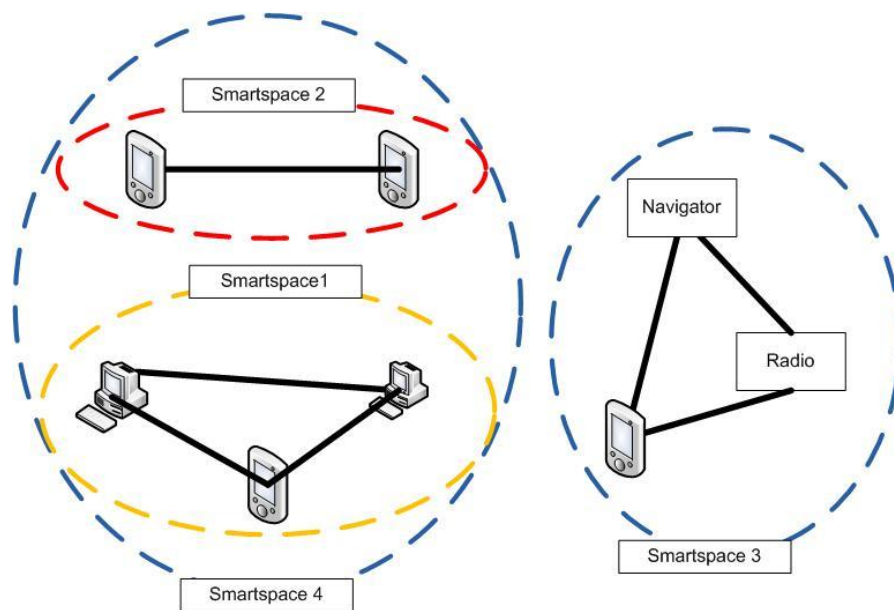


Figure 3.2: PECEs Smart Spaces

3.2 Configuration Tool Introduction

3.2.1 Why we need configuration tool and what inside

As describe in first chapter, the main hypothesis of this research is to create a set of development tools which can easy to be used and speed up the development process for smart space application design and development. Is it

possible to design a tool that will aid the application developer in Configuration the system and increase their productivity? In particular we assume the following questions:

- How to define device information and context by an easy and clear way?
- How to set up security chains for devices by a easy and clear way?
- How to build up the smart space and configure role specification by using the details of device and context?
- How to create service which device support and which can be used by other remote devices?

For answering the above research questions, the main contributions are listed below:

- Configuration tool Architecture
- Device definition tool to define the information of devices
- Ontology Instantiation tool to configure the context of device and smart spaces
- Security tool to set the key and certificate for devices
- Service definition tool to create services and related files for running them remotely
- Role specification tool to define smart spaces and roles
- Create executable device instance which can be run in testing tool

All these contributions will describe in the following sections.

3.2.2 Overview of Configuration Tool

The Configuration Tool provides graphical user interface to configure the virtual devices which will be used for testing a specific application. The Configuration Tool enables application developers to specify the device properties and initial context information during the device configuration. Keys and certificates are used by the security components of the PECES middleware (asymmetric and symmetric cryptography) can be configured by the Configuration Tool as well. The Figure 3.3 shows the architecture of Configuration Tool:

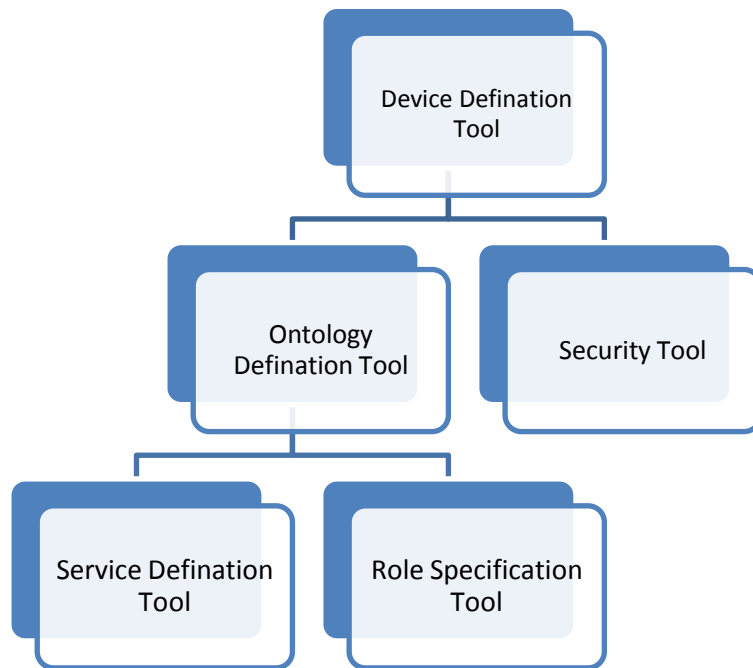


Figure 3.3: PECES Configuration Tool Architecture

The Configuration Tool is responsible for coordinating initial configuration of the device properties, initial/static context information and application and services deployment. The tool provides graphical user interface (GUI) for application developers to specify the device description, initial context elements of devices. The tool also enables the specification of the appropriate keys and certificate. The Configuration Tool gathers initial context information for devices and this information can be used by PECES context ontology functionalities (e.g. eu.peces.middleware.context) at the runtime. This initial context information will be updated by the Modelling Tool where application developers may wish to test different sets of context values to validate any specific application. The device properties and context information of the Configuration Tool GUI representation can be specified through a XML device-description file. The Configuration Tool also loads the necessary device related PECES middleware functionalities to be executed as virtual devices by the Testing Tool. Figure 3.4 shows the relations between different files:

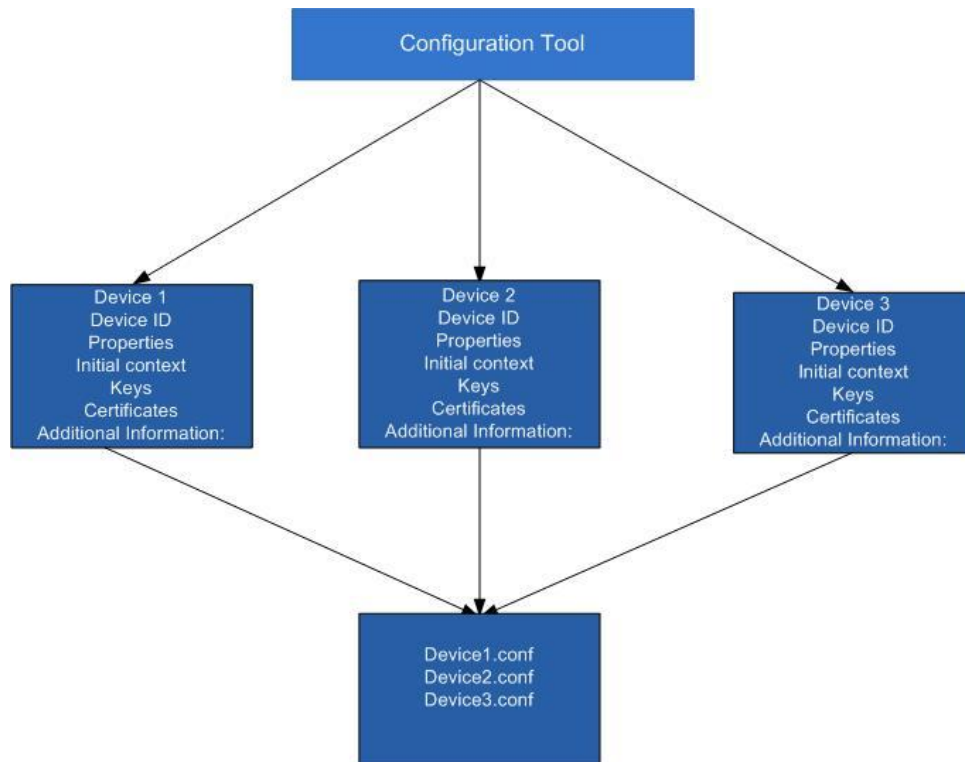


Figure 3.4: PECEC Configuration Tool

The Configuration Tool is responsible for initializing the devices and providing the context information of each device and other additional information. The output of the Configuration Tool is the configuration files which will be used by the Modelling Tool. The Modelling Tool will be discussed in detail in the Chapter 4. Basically, the Modelling Tool will use the Configuration Tool output files as the initial input and provide role specification information, connection and environment information to model the smart space system. The following sub-sections will discuss how the application developers are able to use the Configuration Tool graphical user interface (GUI) to generate initial device configuration which will then be used by the Modelling Tool.

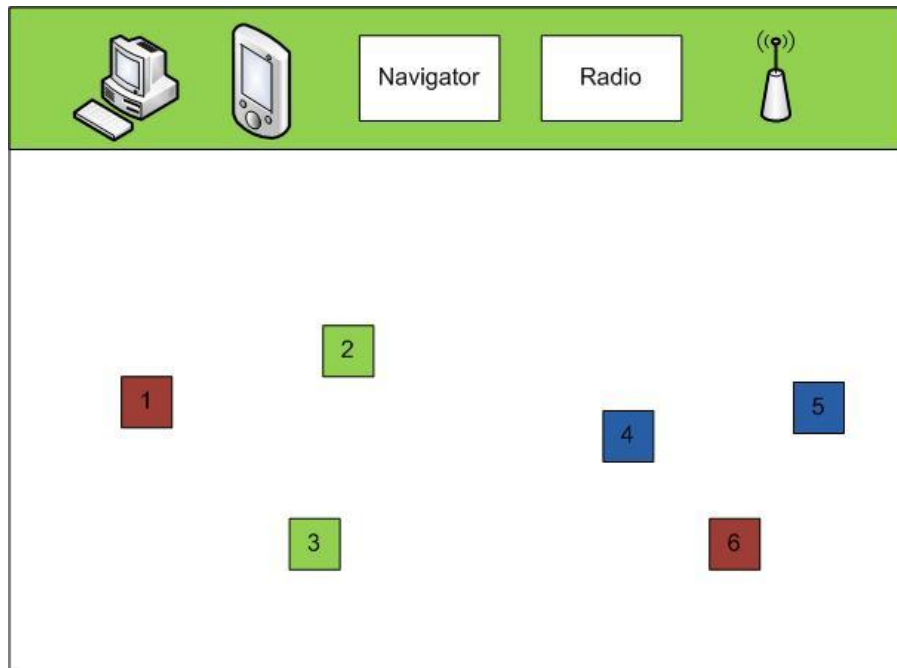


Figure 3.5: Configuration Tool GUI Prototype

The Configuration Tool GUI has icons related to the devices are used in the PECES prototype applications. Developers drag necessary devices which are necessary to make any specific application and place them in the Configuration Tool workspace. The devices have included some properties such as communication features by default but application developers are able to edit them if necessary. Developers have to provide specific ID to each device and once the ID is provided the device appears on the workspace as shown in figure 3.5.

3.2.3 Support for Role assignment architecture

Configuration Tool generates the basic information and provides some configuration for supporting Role assignment architecture.

- The PECES Ontology Instantiation Tool enables the application developer to instantiate the devices. This tool supports all PECES ontologies as well as other custom ontologies which application developers may wish to use for their application. All these context information will be gathered by Context Management layer in Role assignment architecture and can be used as initial data to make accessible to the generic role assignment layer.

- Role Specification Definition Tool provides an interface where developers can define the different rules that the application will use to dynamically form groups of collaborative devices. These rules can be used in role assignment layer to evaluate conditions.

Detail of Ontology Instantiation Tool and Role Specification Definition Tool will describe in the following section.

3.3 Device Definition Tool

3.3.1 Device Definition Tool Prototype

The Configuration Tool GUI provides several device icons in the toolbar where application developers can use very simple drag and drop method to place the required devices in the Configuration Tool workspace. Device properties are already defined in the device icon and application developers may use the device default information or can modify by double clicking on device.

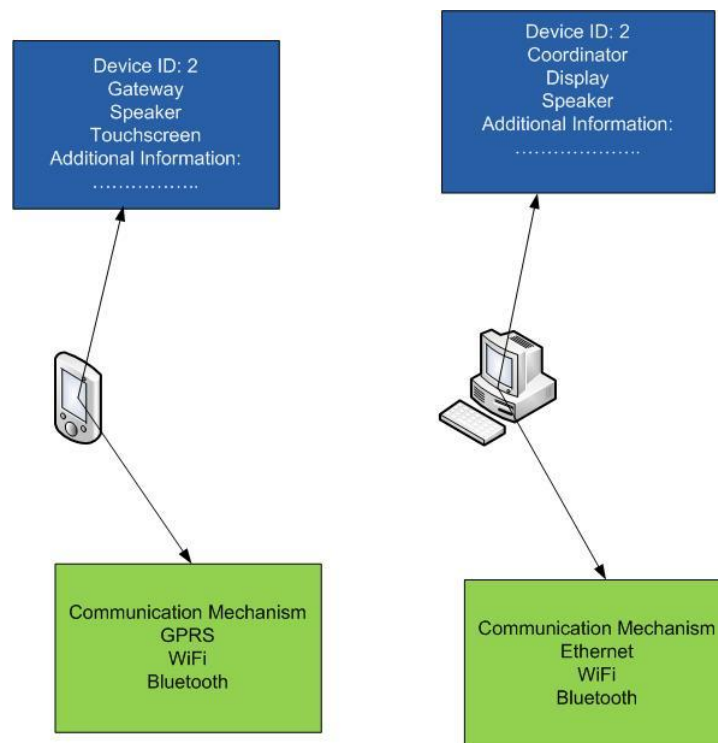


Figure 3.6: Configuration Tool Device Properties

After placing the device in the workspace, device ID will be automatically generated according to the order of the placement (e.g first device placed in the workspace will be give to 1, next device will be give 2 and so on). Application developers may be able to change the device ID by double clicking on the device. Application developers should select the device PECES functionality choosing whether the device implements the coordinator, gateway or member. After application developers make this selection, the device will appear with the selected device ID and corresponding device functionalities colour (e.g. coordinator = purple; gateway = blue; member = green) as shown in Figure 3.5.

Some of the device description information such as communication mechanism will only be used by the Configuration Tool and other information such as coordinator, gateway, member and device ID will also be used by the Modelling Tool. For example, the type of communication links (Figure 3.6) available will be necessary for determining the PECES communication plug-ins to be deployed on the device. The devices will be attached with the corresponding application which includes middleware component and services. If a device in the tool bar is used in more than one application, application developers should have an option to select the required applications from the list from all possible applications.

3.3.2 Device Definition Tool Design

This tool provides a graphical user interface (GUI) for application developers to specify the device description as discussed in above. The PECES Device Definition Tool can be used to define BASE/PECES middleware communication plugins such as IP, Bluetooth, ZigBee (e.g. MxIPBroadcastTransceiver, MxIPMulticastTransceiver, EmulationTransceiver), and device functionalities (e.g. Coordinator, Gateway, Coordinator&Gateway, Member) and also device names. Developers can choose the EmulationTransceiver plugin if they wish to emulate their application. All smart space applications should define one device with coordinator functionality which is responsible of the coordination of the smart space and a gateway device is necessary if the devices in the smart space should access other smart space services.

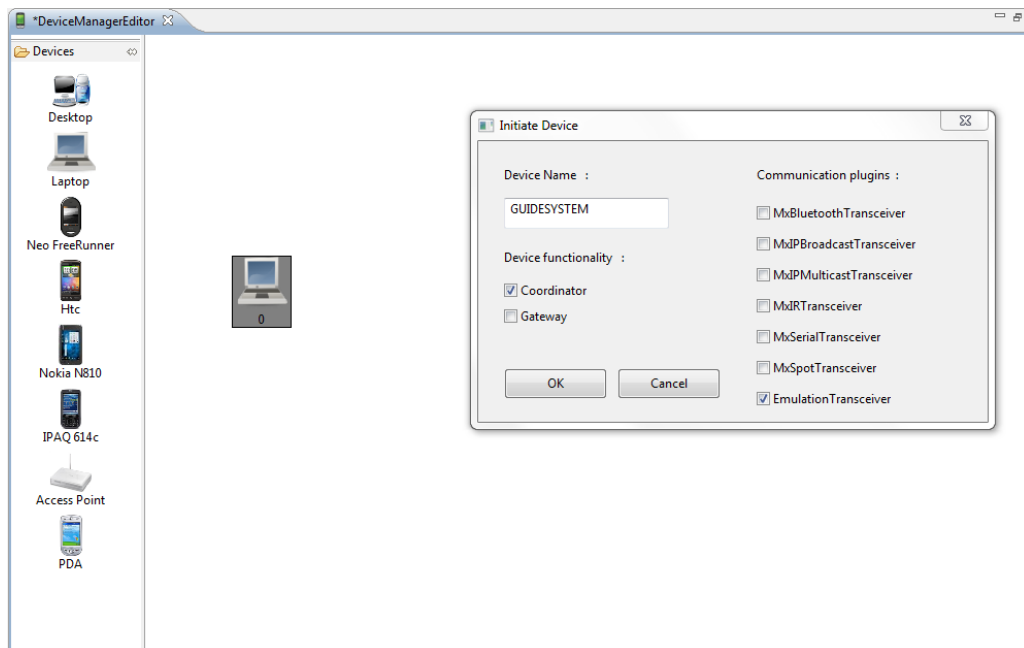


Figure 3.7: Screenshot of the Peces Device Definition Tool with Device Functionalities

The devices can be selected and placed in the Editor area of the tool and necessary functionalities can be defined by right clicking on the devices. After defining the device functionality, different colours will be shown according to the selected device functionality (e.g., a coordinator is red).

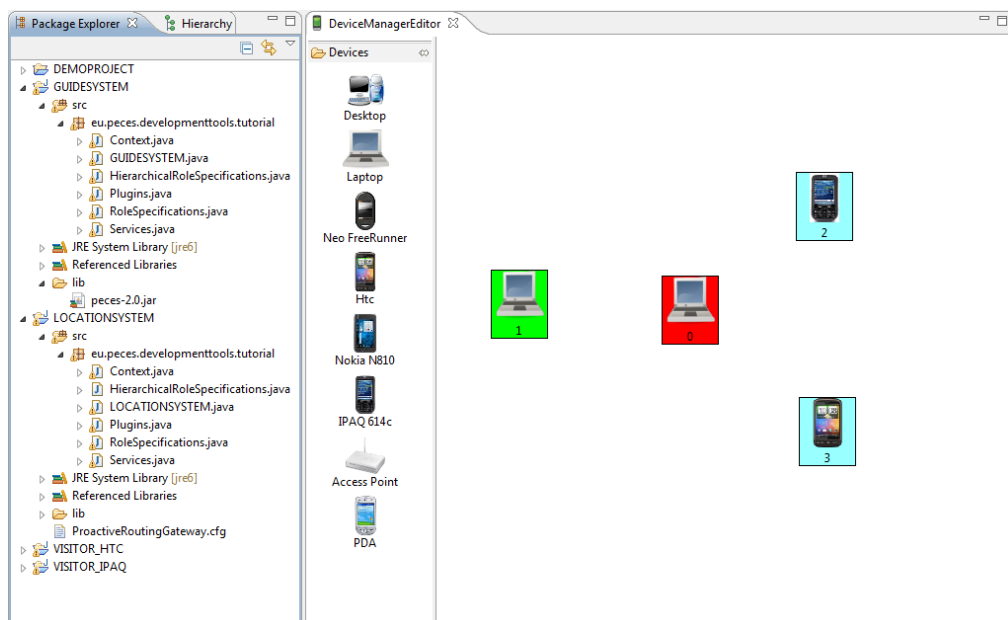


Figure 3.8: Screenshot of the Peces Device Definition Tool

Figure 3.8 shows an example application in which four devices are defined (GUIDESYSTEM, LOCATIONSYSTEM, VISITOR_IPAQ, VISITOR_HTC). The GUIDESYSTEM is defined as the coordinator of the smart space (shown in red)

and LOCATIONSYSTEM is defined as a gateway device (shown in green). Two member devices are VISITOR_IPAQ and VISITOR_HTC and those devices are shown in blue in Figure 3.8.

The screenshot also shows four different Java projects which are automatically generated by the tool namely GUIDESYSTEM, LOCATIONSYSTEM, VISITOR_IPAQ and VISITOR_HTC. These Java projects include PECES middleware libraries (peces-2.0.jar) and necessary Java files under the src folder. These Java projects are configured with the additional PECES Nature which automatically generates Java files from the context definition (*.pctx) file.

After placing the selected devices in the workspace, device IDs are automatically generated according to the order of the placement (e.g. first device placed in the workspace will be given an ID of 0, the next device will be given ID 1 and so on). Application developers may change the device name and device communication features as well as device functionalities such as coordinator, gateway and member but the device ID cannot be changed. The device related configuration details are recorded in project.xml file. *The project.xml file generated in this example application is presented in Appendix 1.*

3.4 Ontology Definition Tool

3.4.1 Ontology Definition Tool Prototype

The PECES context ontologies are composed by the *SmartSpace*, *Measurement*, *Device profile*, *User Profile* and *Event* ontologies, as defined in Context Ontology and Query Specification. The document clearly explains the dependencies among them, as well as the external ontologies which provide a basis for the PECES concepts and properties. The core ontology for representing contextual information of a smart space is the *SmartSpace* ontology. The basic concepts to model the contextual information of a smart space are *Device*, *Context*, *Location* and *Service*.

The *Device profile* ontology provides vocabularies to model specification of devices inside smart spaces. There are three categories of devices defined in the PECES prototype application which are *PECESEmbeddedDevice*, *Accessory* and *SensorDevice*. *PECESEmbeddedDevice* represents those

embedded devices that deploy the PECES middleware. There are three categories of embedded devices according to their role inside a smart space, namely gateway, coordinator and member. In order to specify which kind of accessories an embedded device has, the property *hasAccessory* can be used to link a *PECESEmbeddedDevice* instance to an *Accessory* instance. *Accessory* instances are *Keyboard*, *Touch Screen*, *Speaker*, *Screen* and *Microphone*. In addition to this, *SensorDevice* has two sub-concepts: *Detector* and *MeasuringSensor*. A *MeasuringSensor* instance represents a sensor which can measure a *measurement* such as *light*, *noise*, *temperature*, etc.

The Configuration Tool provides GUI mechanism to define static context information relevant to the device and this information is used by the PECES middleware context components during the model execution. For example, application developers may need to specify a device's mobility-related information, such as whether the device is *Stationary* or *Non_Stationary*. In some cases, the application developers should provide some sensor context values (light, noise and temperature, and the location of the measurements) to configure a device.

Smart space mobility-related information (*StationarySmartSpace*, *Non_StationarySmartSpace*) can only be specified by the Modelling Tool, as smart spaces are formed based on role specification. Application developers will be able provide context information by double clicking on any device in the Configuration Tool workspace. The list of context properties related to specific device will appear on a new window where application developer can enter values or select possible option (e.g. *Non_Stationary*, *Stationary*).

The Configuration Tool only provides suitable static/initial value for context information and the Testing Tool injects this information to the PECES context functionalities. Nevertheless, application developers may be interested in providing dynamic context information, in order to simulate contextual changes and observe the behaviour of the applications under new context values. This kind of context dynamics can be defined by the Modelling Tool.

3.4.2 Ontology Definition Tool Design

This tool implements the features discussed in above and provides a user interface for static context properties. The Peces Ontology Instantiation Tool enables the application developer to instantiate the devices. This tool supports all PECES ontologies (e.g., <http://www.ict-peces.eu/ont/device.owl>) as well as other custom ontologies (e.g., <http://www.daml.org/services/owl-s/1.1/Service.owl>) which application developers may wish to use for their application (Figure 3.8). The Ontology Instantiation Tool automatically loads the participating device name and its assigned functionality information from the project.xml file which was generated by the Peces Device Definition tool. The Peces Ontology Instantiation Tool provides GUI where application developers can add instances and link context properties. When the instantiation process is completed, the tool creates a project.owl file (*a sample file is available in Appendix 2*) in the DEMOPROJECT project ConfigurationTool folder and also creates *.pctx files (*a sample file is available in Appendix 3*) which contains the device context information for each device. Those device *.pctx files are placed in the appropriate Java project and once placed in the Java project, the PECES Nature automatically creates necessary Java files for the middleware from the *.pctx files. The *.pctx files are used to provide local context information about the devices.

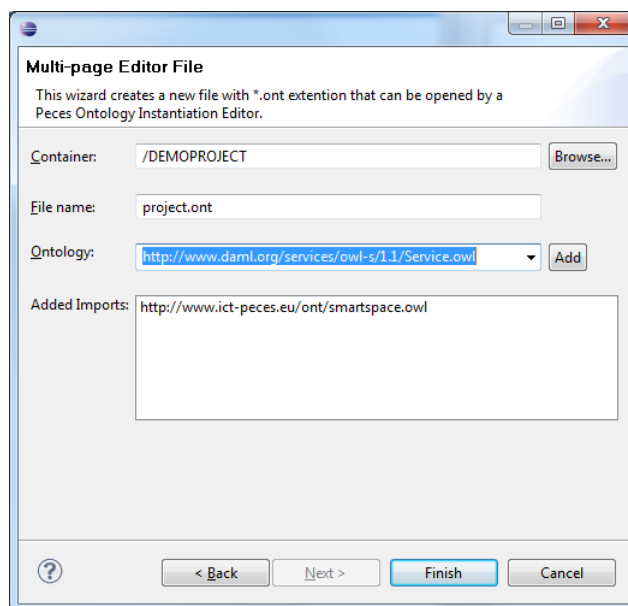


Figure 3.9: Screenshot of the Peces Ontology Instantiation Tool Wizard

Figure 3.10 shows an example ontology instantiation process in which four devices defined (GUIDESYSTEM, LOCATIONSYSTEM, VISITOR_HT and VISITOR_IPAQ) in the Device Definition Tool are automatically loaded by the Ontology Instantiation Tool. Two new services (GuideService and LocationService) are defined here with the Ontology Instantiation Tool. The GUIDESYSTEM is defined to provide the GuideService and the LOCATIONSYSTEM is defined to provide the LocationService. Also the VISITOR_HTC and the VISITOR_IPAQ devices are defined to consume both the GuideService and the LOCATIONSYSTEM device is defined to consume the LocationService.

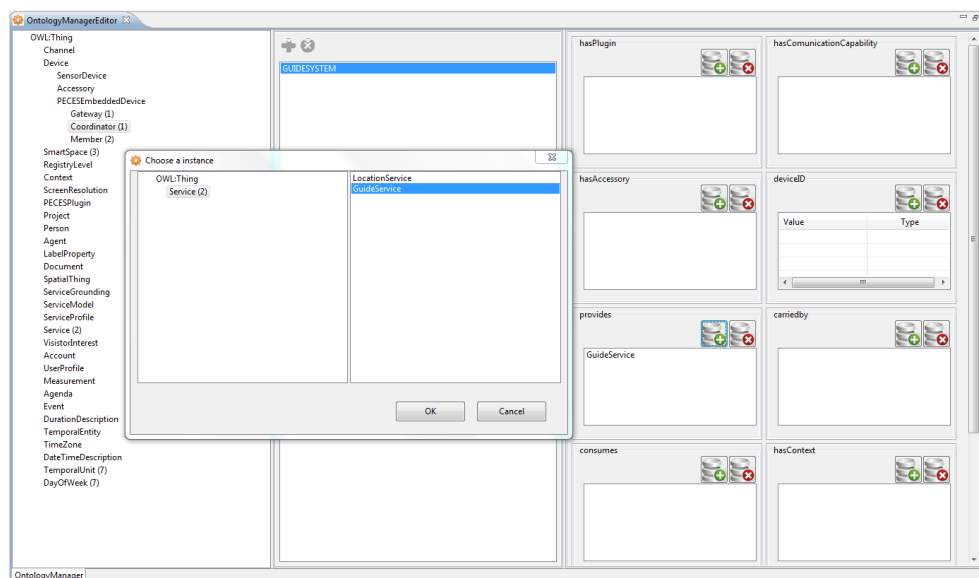


Figure 3.10: Screenshot of the Peces Ontology Instantiation Tool

Up to this point, the developers have configured Java projects for devices without using the features of the PECES secure middleware. The PECES development tools are designed to provide support for both secure and unsecure version of the PECES middleware application development. So far the device projects are configured for unsecure version of the PECES middleware. If the application developers are interested in testing secure middleware applications, then they should use the Peces Security Configuration Tool (Section 3.4) to configure secure middleware Java projects and generate necessary keys and certificates for the devices. Otherwise developers can skip the Peces Security Configuration Tool and move on to the Peces Service Definition Tool (Section 3.5) to continue application development with the unsecure PECES middleware.

3.5 Security Tool

As described in Chapter 2 (Secure Concept), the PECES middleware relies on asymmetric and symmetric cryptography that requires the availability of keys. In the case of symmetric methods, the keys are available only to a particular set of devices which may use this key to ensure different security goals, such as authenticity with respect to the set of devices that shares the key. In the case of asymmetric approaches, the key actually consists of a pair consisting of a public part (public key) and a private part (private key).

Certificates are used to express trust between set of devices by pointing to the certificate at a particular level and then verifying whether a particular certificate belongs to that sub-tree by recursively validating the certificate chain from bottom to top. For this purpose, each device is equipped with a number of certificates to denote the sets of devices that are trusted. These certificates do not need to refer to individual devices and they only have to refer to higher-level entities.

The PECES middleware makes use of the X.509 standard for encryption purposes. This standard defines a common format for certificates which enables the use of existing tools to generate keys and certificates. PECES middleware will use the implementations provided by the OpenSSL library. As a result, the Configuration Tool integrates the OpenSSL toolkit to enable application developers to generate keys for smart space devices which are supported by the PECES middleware security components.

The Configuration Tool provides mechanism to generate public and private keys for each device by using OpenSSL tool. It also provides mechanism to generate certificates. The OpenSSL tool is an open source toolkit implementing the secure sockets layer and transport layer security protocols as well as a full strength general purpose cryptography library. Application developers are able to generate asymmetric and symmetric cryptography keys and certificates for their devices. The OpenSSL toolkit includes a command-line tool for using the various cryptography functions of OpenSSL's cryptography library from the shell. It can be used for creation of RSA, DH and DSA key parameters and creation of X.509 certificates.

The PECES middleware uses the OpenSSL library to create necessary certificates and keys. As a result, the Security Configuration Tool integrates the OpenSSL toolkit to enable application developers to generate keys and certificates for smart space applications. The Security Configuration Tool provides an interface to gather necessary information for root certificate, intermediate certificate (trust chain) and client certificate. The necessary information gathered from the Java interface is passed to the OpenSSL command line interface with the use of AutoIT script files. The AutoIT “Send” command is used to send information.

3.5.1 Root Certificate Configuration

Figure 3.11 shows the interface needed to generate a root certificate. Developers should first generate a root certificate and then are able to generate necessary trust chain and client certificates.

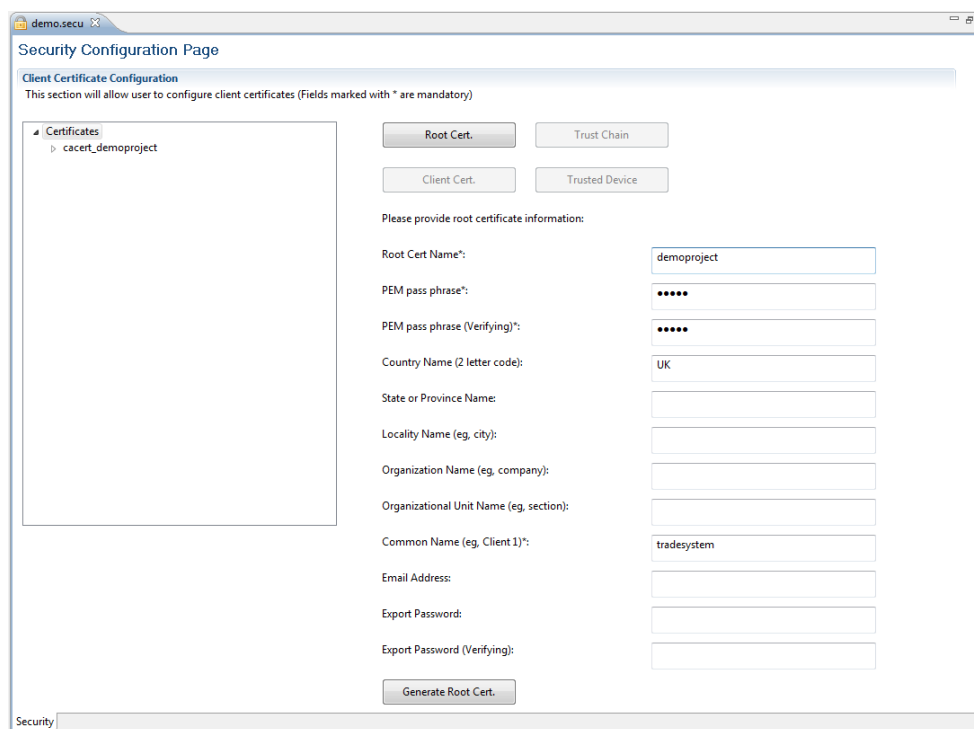


Figure 3.11: Screenshot of the Security Configuration Tool – Root Certificate Creation

3.5.2 Intermediate Certificate Configuration

Once the root certificate is generated successfully, the name of the security root certificate appears as a tree structure in the root Certificate section. To generate the first trust chain, developers should select the root certificate and then click on the “Trust Chain” button which provides an interface for trust chain

configuration. More trust chains can be added as required for the application development.

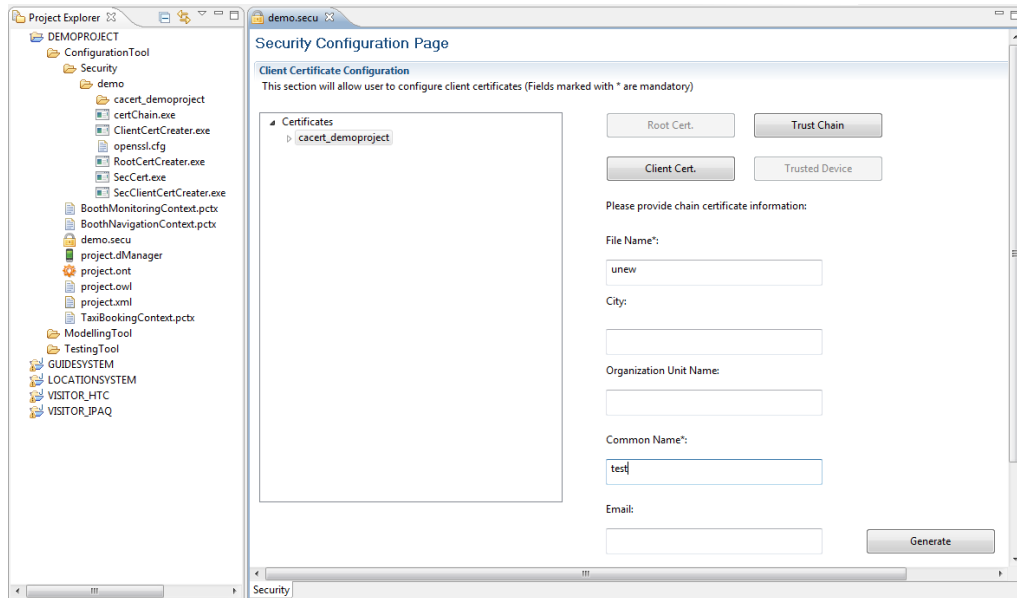


Figure 3.12: Screenshot of the Security Configuration Tool – Trust Chain Creation

3.5.3 Client Certificate Configuration

Once necessary certificate chains are created, they appear as trees in the Certificates area. To generate a Client certificate, first, developers must select the appropriate trust chain in the tree, and then click on the “Client. Cert” button to generate client certificate. The new interface provides feature to select the device for client certificate configuration. For example, here we select the GUIDESYSTEM as the device where the certificate will be deployed. When the process is completed, all necessary root and intermediate certificates are deployed in the “full” folder (full trust) in the certificate folder and keys and client certificates are also deployed in the certificate folder as shown in the figure below (certificate-demo folder in the GUIDESYSTEM Java project).

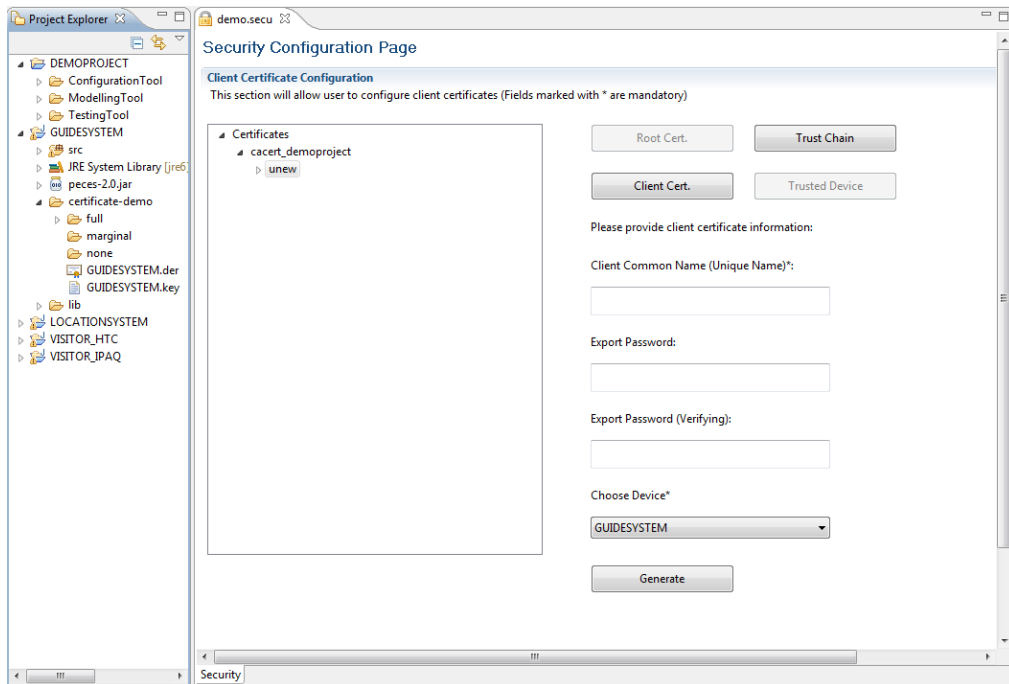


Figure 3.13: Screenshot of the Security Configuration Tool – Certificate Creation

3.5.4 Trusted Device Configuration

The tool also provides a mechanism for a device to deploy certificates to other devices which are to be trusted. For example, the GUIDESYSTEM device can specify that the LOCATIONSYSTEM is to be trusted by copying the necessary certificates to the device. Figure 3.14 shows the trusted device configuration page

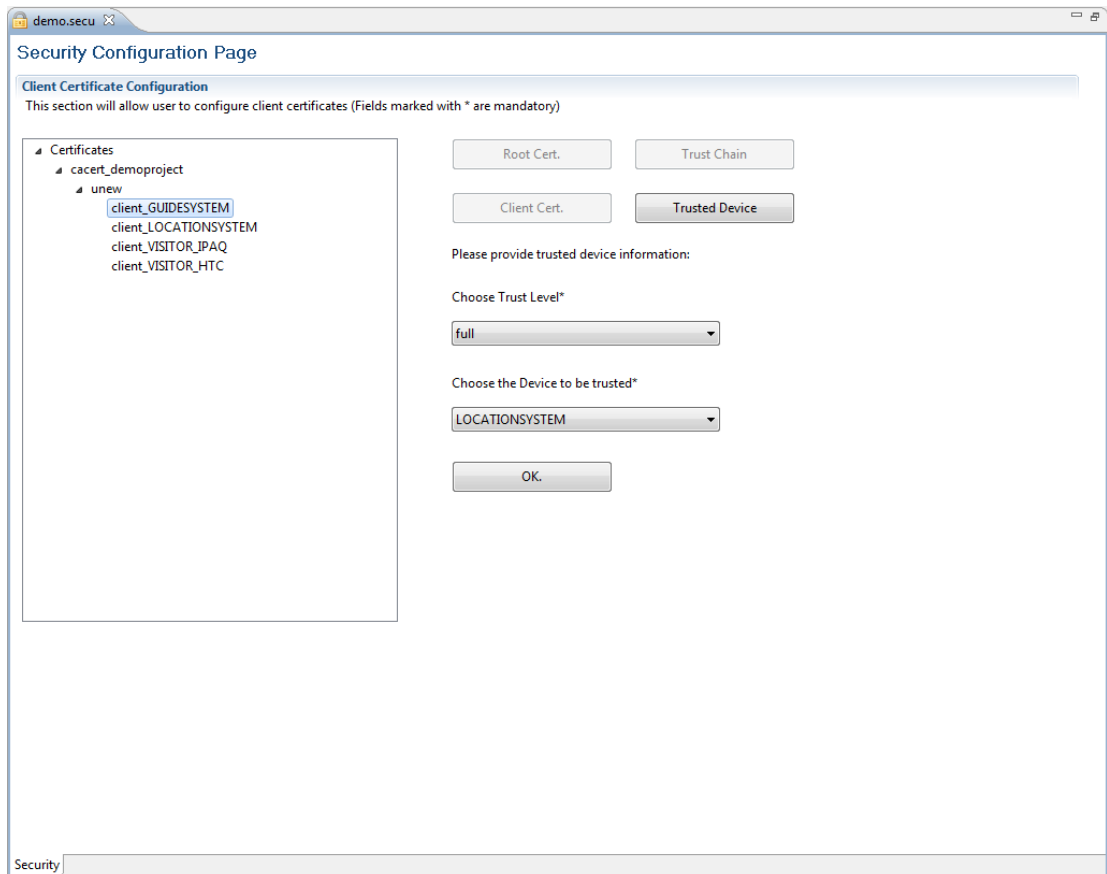


Figure 3.14: Screenshot of the Security Configuration Tool – Trusted Device Selection

3.6 Service Definition Tool

The PECES Service Definition tool provides a simple interface to the developers that allows the automatic generation of all the code needed to instantiate and make use of a PECES-based service.

When the developer decides to make use of the PECES Service Definition tool to define a service, a window shows a list of all the services that have been defined with the PECES Ontology Instantiation tool. The developer can then simply choose the service to be defined.

Once the selection is performed, the main window of the PECES Service Definition tool appears on the IDE, showing the following options to configure the service:

- Device: the device that will be implementing this service. The list of possible devices is shown to the developer, based on the previous work process. The PECES Service Definition Tool automatically infers which is

the possible candidate, but the developer can always change this default configuration.

- Scope: determines at which scope the service will be published. According to the PECES Communiation Mechanisms and Registry Interface Specification, the possible scopes are “Device” (available only to clients on the same device), “Space” (available to devices within the same smart space) and “Internet” (available to all smart spaces).
- Implemented functions: the remaining menus of the PECES Service Definition tool permit the developer to define the interface that the service will offer to its clients (i.e. the functions that will be available to them). This definitions follow a format that is similar to any Java function. It means, the final function will have the following structure: [Returns] [Name]([Parameters]). For instance, “void getGuideLocation()”

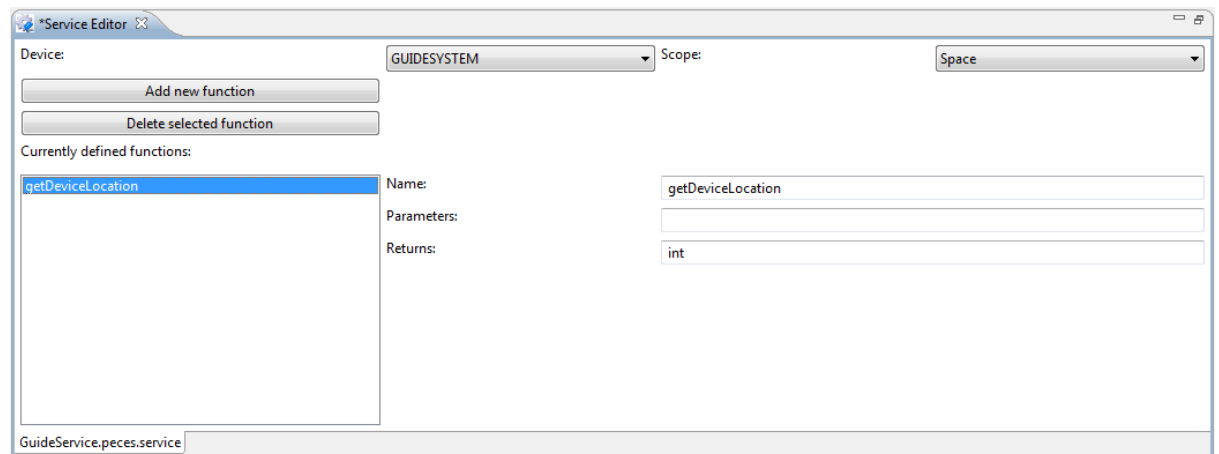


Figure 3.15: Screenshot of the Service Definition Tool

After saving changes, all necessary code is generated where needed:

- [name_of_service].peces.service: this file is generated in the “Modelling Tool” folder of the PECES project, and contains all necessary information to modify the service once it has been created. Any modification to the service must be performed by editing this file.
- Services.java: this file is automatically generated on each device’s project instantiating a service. This code performs the initialization of all the proper services, and is run as part of the application initialization.
- [name_of_service]Service.java: this file is automatically generated and includes an empty body for the service. Several “TODO” annotations

indicate the places where the implementation of the services needs to be added.

- Service stubs and proxies: in addition, several .java files are generated with all the code necessary to make PECES able to access and work with the service.

3.7 Role Specification Definition Tool

The PECES Role Specification tool provides an interface where developers can define the different rules that the application uses to dynamically form groups of collaborative devices.

This tool targets specifically part of the context definition task. That task describes how the role specifications need to be deployed in the devices in order to trigger the smart space formation process. The PECES Role Specification Definition tool assists the developers in the design of those Role Specifications that defines the grouping process.

In a PECES project, these rules are written essentially as constrained queries over the context properties of the devices. For that reason, the PECES Role Specification Definition tool loads the results of the PECES Ontology Instantiation tool, showing on a tree-shaped diagram all the devices that have been defined in the project, and their properties (upon which the rules will be defined). The process to define a new Role Specification is as follows:

- 1) Select the device that instantiates the Role Specification. This device must be a coordinator. For these reason, a combo box with all coordinators defined for the project is presented, where developers can choose the proper one.
- 2) Select the scope of the Role Specification. According to the PECES Communication Mechanisms and Registry Interface Specification, there are three possible scopes: "Device" (available only to clients on the same device), "Space" (available to devices within the same smart space) and "Internet" (available to all smart spaces).
- 3) Select the member's minimum trust level. In the final version of the PECES middleware, it is possible to include certificates-based security and trust concepts in PECES applications. In that case, the developer are able to

select the minimum trust level that all members must fulfill with the coordinator in order to become members of the smartspace (None, Marginal or Full). If no security concepts are being applied, the “Don’t apply” option should be selected.

- 4) Append Rulesets to the Role Specification. A Ruleset is a constrained query over the context properties of a device. A device fulfills a Ruleset when ALL the conditions defined there are fulfilled (AND conditions). On the other hand, a device fulfills a Role Specification when at least one of its Rulesets is fulfilled (OR conditions). Therefore, by combining several Rulesets in a single Role Specifications, reasonably complex conditions can be applied to the group formation process.

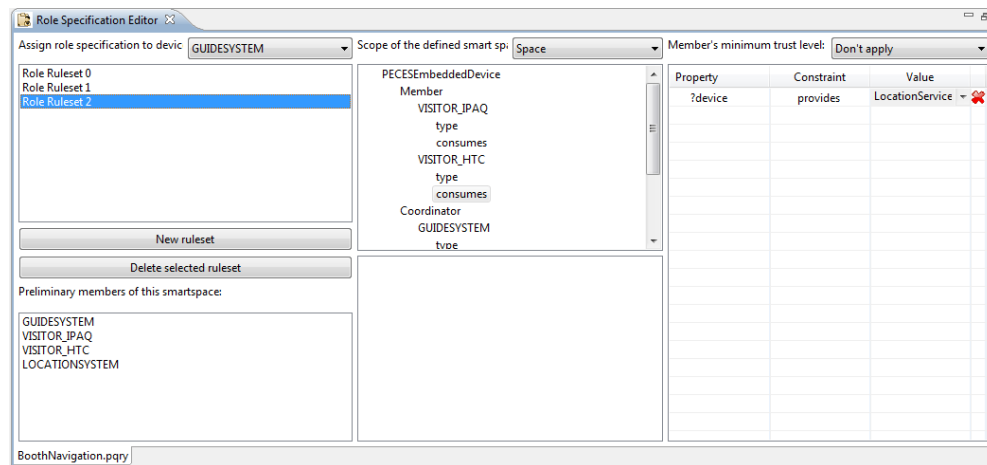


Figure 3.16: Screenshot of the Role Specification Definition tool

When a Ruleset is selected, its definition can be altered using the two tree-shaped property diagrams and the right-hand window editor.

- By double-clicking a property in the devices tree, a constraint over that property is added to the Ruleset. For instance, a constraint “?device provides ?service” would mean that “any device providing any service” fulfills the Ruleset.
- The window showing the defined constraints allow to change the third part of the constraints (“?service” in the example) by any possible value actually defined (possible values are shown in a combobox). For instance, a constraint “?device provides guideService” would mean that “any device providing guideService” fulfills the Ruleset.

- If the third part of a constraint is left undefined (i.e. beginning with an interrogation mark), it will appear in the variables tree. By appending a property of that variable to the Ruleset, composed constraints can be designed. For instance, two constraints – “?device provides ?service” and “?service isConsumedBy the GUIDESYSTEM” – would mean that “any device providing any service consumed by the GUIDESYSTEM device” fulfills the Ruleset.

During the whole definition process, the bottom-left window with the title “Preliminary members of this smartspace” shows a prediction of the devices that fulfill the Role Specification, according to the static and initial context properties of all the devices defined within the project (note that as the applications run, their context may change, thus dynamically changing the members list of each smart space).

By saving the Role Specification definition, the tool automatically generates all necessary code in the required projects to define and instantiate it using the middleware:

- [name_of_role_specification].pqry: this file is generated in the “Modelling Tool” folder of the PECES project, and contains all necessary information to modify the role specification once it has been created. Any modification to the role specification must be performed by editing this file.
- RoleSpecifications.java: this file is automatically generated on each device’s project instantiating a role specification. This code performs the initialization of all the proper role specifications, and is run as part of the application initialization.
- [name_of_role_specification].java: this file is automatically generated and includes the actual code defining the described role specification. It is used during the instantiation made in RoleSpecifications.java

3.8 Hierarchical Role Specification Tool

In the final version of the PECES middleware, it is possible to define smartspaces hierarchically, as unions of previously instantiated role specifications.

That task considered smart space formation based on Role Specifications. This is the basic method, but advanced version of the PECES middleware allows the definition of hierarchical smart spaces, defined as the union of smaller smart spaces. Therefore, it was convenient to introduce a new tool in the set of development tools, assisting the developers in the use of this new feature.

The PECES Hierarchical Role Specification tool provides an easy method to create all the code necessary to instantiate this kind of “composed” smart spaces.

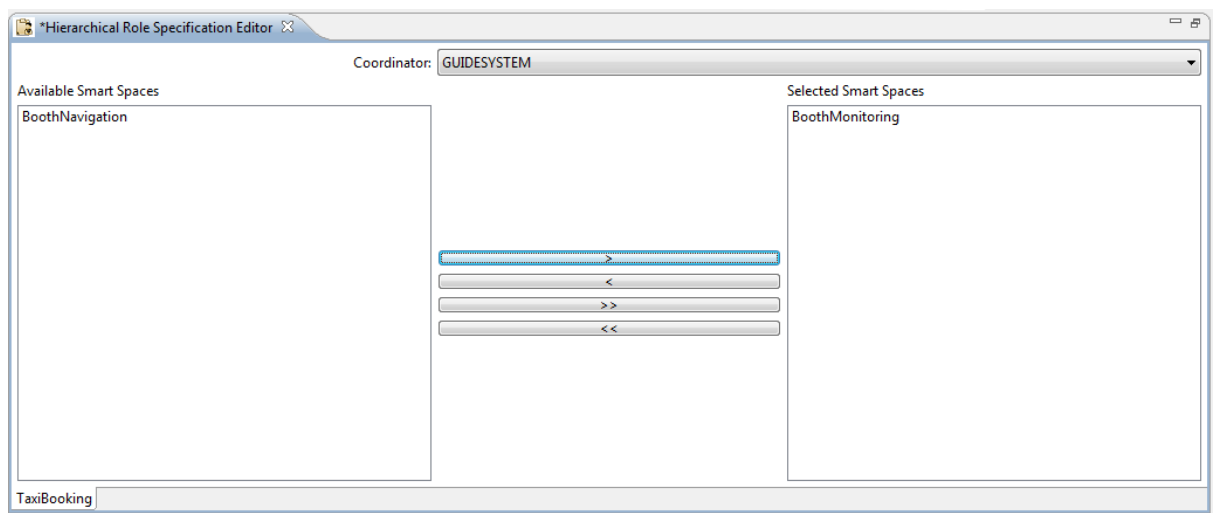


Figure 3.17: Screenshot of the Hierarchical Role Specification Definition Tool

The tool presents a list with all the smart spaces included in the project. The following steps must be taken in order to define a hierarchical smart space:

- 1) Choose which coordinator will be in charge of instantiating the hierarchical smart space. A combo box listing all coordinators defined in the project is shown to the developer.
- 2) Choose which smart spaces will compose the hierarchical smart space. The developer can easily compose the list of “selected smart spaces” by using the proper buttons.

By saving the Hierarchical Role Specification, the following files are created:

- [name_of_hierarchical_role_specification].phqry: this file is generated in the “Modelling Tool” folder of the PECES project, and contains all necessary information to modify the hierarchical role specification once it has been created. Any modification to the role specification must be performed by editing this file.

- HierarchicalRoleSpecifications.java: this file is automatically generated on each device's project instantiating a role specification. This code performs the initialization of all the proper hierarchical role specifications. This code is not automatically called during initialization, since its execution only makes sense under certain conditions during application runtime. Therefore, it is responsibility of the developer to decide when to instantiate the hierarchical role specifications, based on the application logic.

3.9 Cooperation with Modelling Tool

As discussed above, the Configuration Tool is responsible for the initial configuration of devices, initial context, keys and certificates and application and services. This tool provides necessary device configuration information and other device related information to the Modelling Tool. The Modelling Tool uses this information and defines role specification, environment, and connection information to provide a ready-to-run application model to the Testing Tool. The Configuration Tool output is device description XML files which is generated from the GUI interface description of the Configuration Tool. Using the Configuration Tool output XML files, the Modelling Tool can be able to generate a GUI setup with configured devices in the Modelling Tool workspace to model the smart space application by providing necessary additional information.

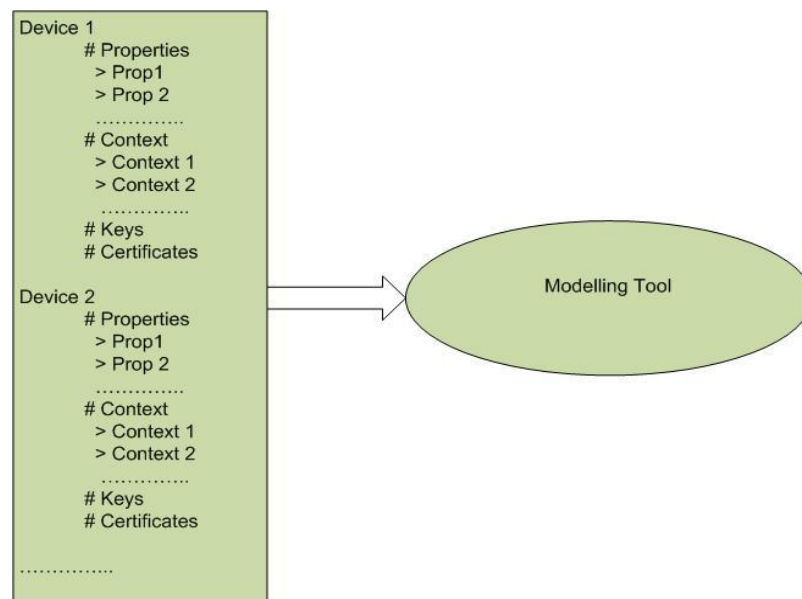


Figure 3.18: Cooperation with the Modelling Tool

Chapter 4: Development Tool Design: Modelling Tools

In this Chapter we will discuss the Modelling Tool which get the information from configuration tool and is used to define environment and event to provide a ready-to-run application model to the Testing Tool. Alberto from ETRA, Spain, gave lots contributions to help build Modelling tools. First of all, we will explain the architecture and structure of modelling tool and how the modelling tool work with other tools in Section 4.1 In Section 4.2 we will describe the Event Editor which can define several event include dynamic context change and connection. We will also show how to define the sequence of event by using Event Diagram Editor in Section 4.3. We will end with a brief summary of this chapter in Section 4.4.

4.1 Overview

4.1.1 Why we need modelling tool and what inside

The main task of the Modelling Tool is to support application developers in specifying the environment, the inter-entity binding and the event definitions which describe the dynamics of the model. The Modelling Tool also allows the specification of the communication mechanism (routing) between the individual devices such as requirements on encryption and the type of data exchanged between the devices. Basically, at this point the developer can start with the definition of the environment where the application or service can be run and can be validated. The result at the end of the Modelling Tool section will be, as its name says, a model simulating the real life circumstances for the applications and the services. With this, developers can gain a feedback of their design even before the first real deployment of their software. The tool is responsible for creating “ready to run” instances and also holding the information of the model as a global unity in order to be able to provide this information to the Testing Tool which is responsible for the execution of the previously defined information.

Is it possible to design a tool that will aid the application developer in modelling the discrete events and sequence and increase their productivity? In particular we answer the following questions:

- How modelling tool work with configuration tool?
- How to define single event and how many different event types can be simulated by development tool?
- How to define an executable event sequence by a simple and clear way?

For answering the above research questions, the main contributions are listed below:

- Modelling Tool Architecture
- Event definition tool to define single event
- Event Diagram Editor to link events into a sequence which can be run in Testing tool

All these contributions will be described in the following sections.

4.1.2 Introduction of Modelling Tool

The Modelling Tool provides a graphic user interface (GUI), and the necessary tools to transform the model in XML descriptions in a seamless way to the users. This feature has to be also presented at all tools in order to make it easier for the developer to create the needed definitions. The ontological syntax is an XML-like syntax. There are several previously generated definitions which are available for the developers. Besides that, the tool has to provide the ability to instantiate these ontologies filling them with the application/service specific information. These instantiations are application/service-specific descriptions of the developer's model. In the current view the base device level definitions are defined at the Configuration Tool, while the smart space and inter smart space-related definitions can be defined by this tool. The Modelling Tool has to provide an environment responsible for creation of these definitions. The next important thing is the definition of the simulation scenario which can be done by defining dynamics that are taking place at a certain time. These dynamics are presented as event-timestamp pairs. The event defines exactly what has to be changed, while the timestamp defines when the event is fired. Further sections (Dynamics Modelling) describe in detail which events can be defined and how to do this.

Another important task for this tool is to assemble the components executed in the runtime environment. This includes the association of components and features of the middleware to the devices. For example which device will have a context provisioning module, role assignment engine, etc? For this purpose,

there has to be an assemble perspective within the modelling perspective, where the developer can easily review the main components of the virtual devices and query them in detail. At the end of the modelling step there is a certain number of virtual devices (attached to deployment/build files and their corresponding source code files) and global model information.

Generally the Modelling Tool provides complete device definitions which are ready to be instantiated, connection information and event definitions responsible for the dynamics in the system. This information is passed as a whole to the Testing Tool, which is responsible for instantiation, execution and supervision of the current state and parameters of the system.

Basically there are three perspectives here: the context, communication and events perspective. All three functionalities (context definition, communication definition, event definition) can be presented as different development views, so the developer is able to switch between them and the information available to set and query is present at the corresponding views. In the context perspective, the developer is able to set contextual information; the communication perspective is responsible for defining and setting the connections (communication links) between devices; the event perspective is used for event definition and specification of the dynamics to be simulated during model execution.

An important design aspect must be to keep clear for the developer what can be done with the tools and where these features can be found. An elegant way could be showing a model's graphical presentation always on the UI, while focusing each of the perspective on their correspondent components. As an example, imagine a developer who wants to set the communication links between the different elements of the model. Therefore, the connection view needs to be opened. The developer sees all the device entities and the link connectors between them. Smart Space borders are visible, but become half transparent and inactive, so the developer can concentrate better on the connection related elements. The visualisation of this description can be seen in the sections below (Context Modelling, Communication Modelling).

Figure 4.1 is introducing the logical “inner” architecture of the Modelling Tool. As the first design steps recommend, it must have well-defined interfaces for

communication with the other tools, and a graphical interface for the interaction with the developer. These three interfaces are designed in order to be able to seamlessly communicate with the other tools and, in case of the GUI, to be as much simple and easy-to-use as possible. Beside the three outer interfaces the core of the Modelling Tool implements the sub-modules mentioned above, which are responsible for holding the functionality of the perspectives needed in the exact model definition. The context, communication and event sub-modules mainly communicate with the three outer interfaces, but they will exchange data between each other as well if needed. For example the communication and the event module will exchange data when the initial context is defined and have to be passed to the event generation module in order to define the dynamics for the system during the simulation. The functionality needs to be described in the present document is trivial for the outer interfaces as they have to bridge the current tool with other tools and the developer. The detailed functionality of the core parts in Modelling Tool will be explained in the upcoming sections.

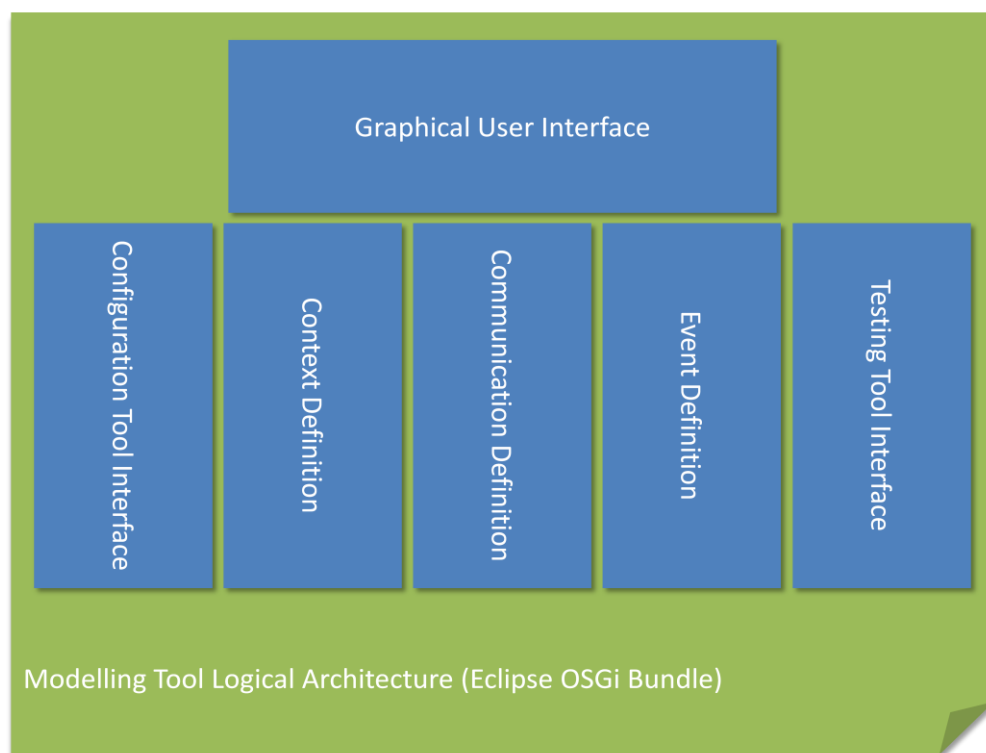


Figure 4.1: Logical Architecture of the Modelling Tool

To understand better the usage of the modelling, a suggested sequence for model definition is shown in Figure 4.2. Note that this is just a recommendation and the developer can switch any time between the views and perspectives and

refine the correspondent parts. The general sequence should be, at first, context extension and global context definition; then the definition of communication links and, as the last step, the definition of the dynamics during the simulation.

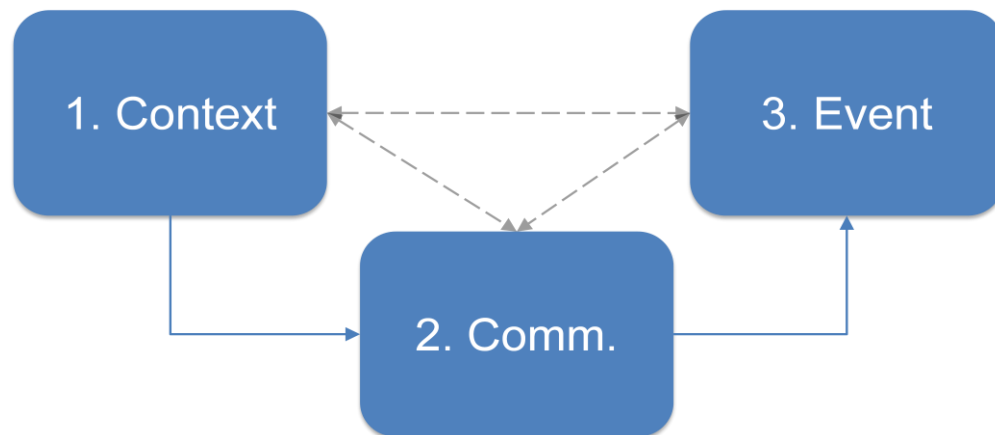


Figure 4.2: Usage for Modelling Tool

4.1.3 Support for Role assignment architecture

Altering a device context needs further considerations than the previous types of events, since development tools should make the access to the actual context descriptions of any device simple to the developer. When the developer finishes, the changes have to be saved and provided to the Testing Tool's event engine. The Event editor manages the context change at certain time stamp. All these change will be gathered by Context Management Layer when this event running in testing tool. It also may change the device status and lead the device is allowed to join or leave smart space.

4.1.4 Cooperation with Configuration Tool

During the sequence of development, Configuration Tool supplies the functionality for the first step of development. This tool provides static configuration information about every instance which includes devices, context, security, role specification and so on. All these information will be passed to Modelling Tool. Then, Modelling Toll will transform these static configurations to a cooperating system model, where multiple instances are connected and communicate with each other.

The following data was defined by Configuration Tool and is provided as input data for the Modelling Tool:

- Context definition of each device.
- Application and service mapping to devices with basic middleware component definitions.
- Keys and certificate mappings for each device when secure communication needs to be added.

4.1.5 Discrete Event Dynamics Modelling

Modelling system dynamics means defining events that happens during the simulation time, thus simulating a dynamically changing environment. This feature is a special one because at this point nearly all the other features used before can be used to change the state of the defined model at a specified timestamp.

At this stage the developer has the model completely defined, containing devices with all necessary contextual information needed on them. The topology and the connections are defined as well. The roles and needed engines are deployed to the devices. The needed middleware components are deployed to the devices (the model defines which components of the middleware have to be made available during runtime on each device).

The definition of a change starts by creating a “New Event” on the modelling GUI interface's “Event perspective” view. Here, the first step is to give a name and description to the event for better understanding of its purpose, and a timestamp stating when to actualize it during the simulation. An important question is whether to use fix real time step for the simulation or to make the simulation steps scalable. In practice, most simulation tools provide a feature for scaling the step of the simulation time. This could be useful for example at big, complex simulations where the developers are not interested in all events and parts of the simulations, but just in a set of these. It is important to mention that these scalable steps just take effect on the event scheduling and not on the whole model so the runtime of the software itself won't be scaled, but just the time of the event triggers. This can be imagined as the developer putting a fine scale at the interesting parts and a larger scale at less interesting parts. For example, a complex simulation which consists of events which are triggered during a longer period and where the developer wants just to inspect a change that normally occurs after an hour. The scale can be raised in order to fire all

not interesting changes faster in order to get almost directly to the interesting point. The importance between dependent changes is present because if the events are speed up the convergence time between dependent changes must still be respected in order to have all the changes in valid circumstances. The convergence time is in most cases very small. Then the tool must indicate a minimum gap between each event so to be sure that the previous change is actualized. This approach can be followed if all the modification delays are at least relatively measureable so to have the exact information about the minimum gap needed to place after each event. This way, the definition of an additional dependency check engine can be excluded.

Summarizing this description in the Modelling Tool the timestamps are implemented as units with no measure, and the scaling of them in comparison with real time can be adjusted in the Testing Tool, during the execution. The main event types provided by the tool are breakpoints, device switch on and off, device context properties change, and communication link creation and destruction. Here the developer has to define a simulation length and after that the events on the timeline.

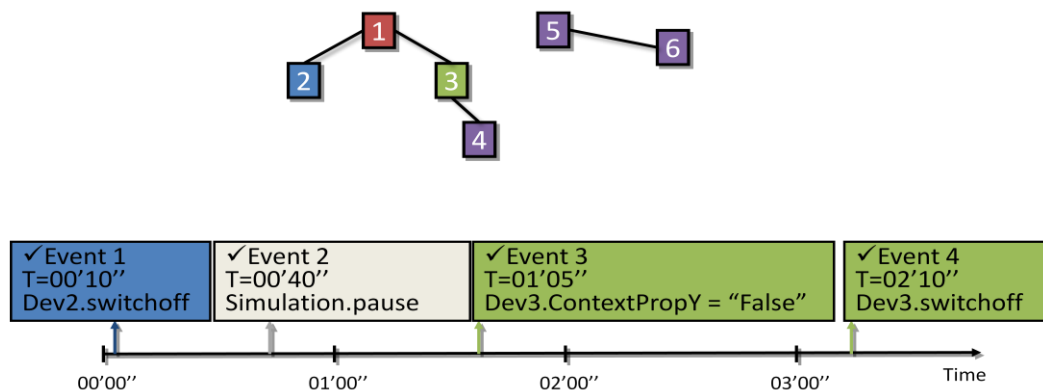


Figure 4.3: Example of Discrete Event Sequence

The following types of discrete events are defined by modelling tool:

a) Convergence Delay

These delays can be imagined as substitutes for usual breakpoints in software debugging, where the execution of the code hangs and the developer can

observe the actual state of the program. In this terminology, a main purpose of a delay is not to break the execution of the model. This feature is a tool for the developer to specify a delay after he/she can investigate whether the changes defined in the events before are present and whether the model at this stage converged to the state the developer is expecting. In this aspect there won't be classical execution breaks (the running of the JVM processes won't be paused). The only purpose of this feature is to have a timestamp from the developer and to dump all the interesting information at this timestamp.

The approach of having a classical breakpoint feature in the PECES development tools has been discarded in a design decision, as it can produce a very complex system. The PECES development tools rely on another terminology of the breakpoint, the so called "convergence delay". With this definition, two timestamps are needed for an event. The first trigger defines the moment when the modifications are performed, while the second one declares a delay for the system to converge and update the modifications. This way, there is no need of hanging the execution of all the JVMs. The Testing Tool gets the required information from the model after the defined delay expires. In this case the classical breakpoint does not be present as an event. Instead of this, the developer will have the information about the model's state after the specified delay in the trace or log file.

b) Device Switch On and Off

Device switching on and off does not mean just hanging the operation of the JVM, but shutting it down completely and later executing it again. The explicit work at this point is also done by the Testing Tool. The Modelling Tool just has to provide the information about when these events happen, and which type of action must be taken. This can be also done in classic XML configuration files which the developers have to fill via the event definition interface. The timestamp indicates when to do the switch and the virtual device's ID identifies the JVM to stop or start. As mentioned before, the Modelling Tool is only in charge of designing the scenario. The Testing Tool - based on this information - monitors the actual state of the virtual devices. Switching on mean starting the JVM with the full environment defined by the Modelling Tool and switching off mean shutting down the corresponding JVM. This needs further features in the

Testing Tool to have an inventory of which JVM process belongs to which device ID, and a simple query feature to get the device ID from the JVMs.

c) Device Context properties Change

Altering a device context needs further considerations than the previous types of events, since development tools should make the access to the actual context descriptions of any device simple to the developer. When the developer finishes, the changes have to be saved and provided to the Testing Tool's event engine. The way of the change definition is not trivial, as two situations can be followed. The first solution is a so called stateless mode. There, the changes defined before this event is not considered for validating the current event. In this case, the developer has more responsibility on defining a valid change, leading to a valid scenario. For example if the developer creates a change where device A is switched off at timestamp 5.0000, and after that defines a connection between devices A and B at 6.0000; this won't generate an error but simply nothing will happen (because device A has been switched off at timestamp 5.0000). Summarizing this explanation in stateless solution multiple context property change events dependency among each other is not checked. A helpful feature of the development tools would be to allow the developer to watch the actual status of the devices upon the change defined in the events timeline, prior to the execution of the model. This is a useful feature but brings more complexity in the defined system. The current view about the implementation of this part is to provide the stateless solution of event definitions where the developer has the responsibility of defining events which make consistent changes to the system at a certain timestamp.

d) Communication Link Creation and Destroy

Describing connection changes is a simpler process. For this purpose the existing UI can be used. The GUI should allow users to choose a specific simulation time, showing the active links at that moment (based on the initial situation and the previously defined events). Once the connection topology appears, the user can delete or add connectors. If connection changes are already added and the developer wants to add or remove a connection with a timestamp beyond already existing modifications then the actual connection topology corresponding to that timestamp has to be shown, and not the initial.

The harder task here corresponds again to the Testing Tool; the Modelling Tool only provides events information, and the event engine has to take care that the corresponding sockets are opened or closed properly.

4.1.6 Cooperation with the Testing Tool

As shown on the Modelling Tool's logical architecture diagram Figure 4.4, the Modelling Tool has an interface for communication with the Testing Tool. This interface is responsible to produce the necessary input from the model definitions done before.

The Modelling Tool has to provide the following information (some extend from Configuration Tool) for the Testing Tool:

- The virtual device instances, which contain:
 - o The application/service code defined by developer
 - o The middleware components used on each device
 - o The keys and all the encryption related information
 - o Role specifications and assignment mechanisms (if available on the devices)
- Event definitions which include the dynamic context change and connection information.
- Event sequence which contain the dynamics through simulation time

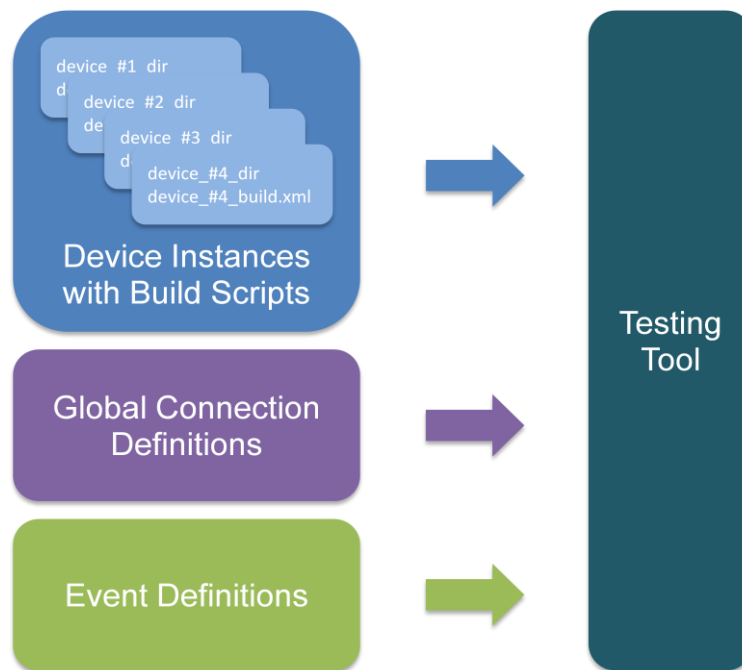


Figure 4.4: Output to Testing Tool

Generally, after using the Modelling Tool to complete the details of the model, the framework has to build complete deployed entity configurations which only have to be compiled by the Testing Tool's instantiation engine in order to be ready to be run in a JVM. This part will be sent to the instantiation engine. For this purpose, additional data about the build mechanism (build file or script) has to be generated for every instance to provide for the testing tool the exact way of the instantiation so the Testing Tool just need to invoke a general instantiation command and the build definitions provides a specialized compilation sequence based on the information in the build file.

The connection information will be simply sent to the connection engine to provide the initial connection topology.

The event definitions are processed by the event engine. Later, during the model execution the event engine has to cooperate with the instantiation and connection engine based on the content of the defined dynamics so to inform the corresponding engines to process them.

The exact deployment functionality is taking place at the end of the modelling section. The developer can do this for each device separately or for all devices at once. After this action is initiated, the Modelling Tool has to assemble the ready to compile packages for each device and make a common package

where the environment related metadata is taking place. This metadata consists of the events and all generic data needed for the testing tool to be able to execute the model. This is implemented as a sidebar where all the virtual devices and the generic data are listed and where the developer can initiate the deployment process. So after this process, when the developer starts the Testing Tool, this data will be an additional input.

4.2 Event Editor

Event Editor is used to edit single discrete event definitions. Type, Contributing Devices, Description and Duration (Delay) can be defined in the wizard and later altered on the Overview Page. The Event Editor is a multipage editor, on the second, Context Page the context of the corresponding device can be changed if the event's type is Device Context Change. On the third page, connection can be defined if the event's type is Connection.

This tool represents the Dynamics Modeling ability of the Development Tools. It comes bundled with Event Diagram Editor detailed in section 4.3 and these two editors form a logical whole in dynamics creation. The following two sections will introduce how these tools help the developer to test the imagined system with artificially produced changes in the context and in the designed topology itself. The benefit of these tools is that the basic algorithms and the business logic of the software and middleware can be validated even before the first real deployment.

The Event Editor is used to edit single event definitions. Type, Contributing Devices, Description and Duration (Delay) can be defined in the wizard and later altered on the Overview Page. The Event Editor is a multipage editor. On the second page, the Context Page the context of the corresponding device can be changed if the event's type is Device Context Change. The third page – Connection Editor Page - is a whole graphical editor wrapped in a multipage editor's page. This editor deals with the definition of the connection related changes. The developer can only switch to this third page if the event's type is Connection Link Change. The developer will work with *.peces.event files. Each file contains the serialized event data itself.

4.2.1 Overview Page

The Overview page is available if the developer wants to alter the event's main parameters after the definition in the wizard. The following screenshot shows one sample event definition page with Type, Contributing Devices and Delay information. This page can be used to modify information and so effectively create new event definitions.

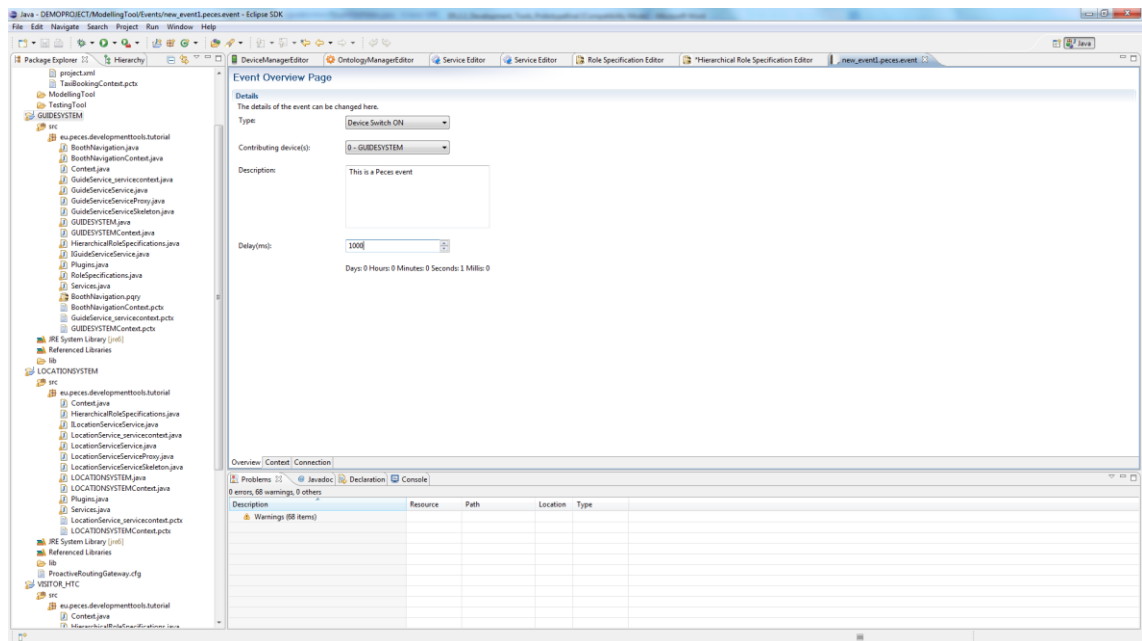


Figure 4.5: Screenshot of the Event Editor Overview Page

4.2.2 Context Page

The Context Page is used when the device type is Device Context Change. Then on the Context Page the properties of the corresponding device can be changed. It consists of two main parts. The left side is a tree view where the device's main properties and the linked instances can be seen. The main properties are selected based on properties visualized in the Ontology Instantiation Tool. On the right side the already defined changes are listed compared to the device's initial context. The change can be easily made using the tree if the developer “right clicks” on the instances or the properties. If the developer “right clicks” on a property, an “Add” context menu will appear and she/he can link instances defined by this property to the device. If the developer clicks on an already listed instance in the tree a “Delete” context menu will appear so the developer can unlink the instance (delete it).

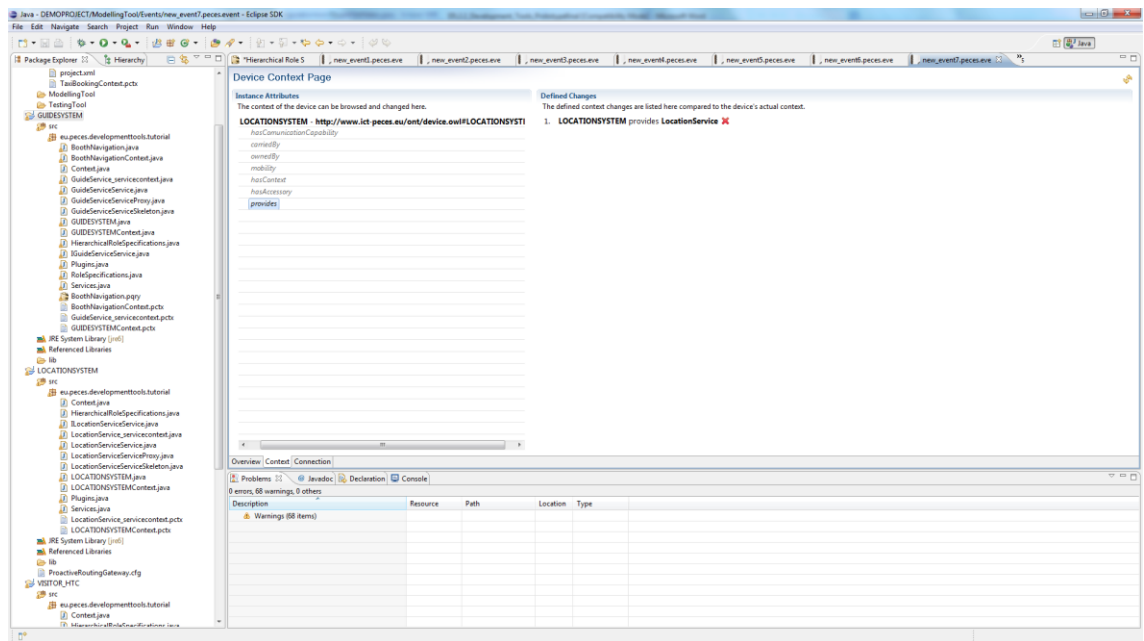


Figure 4.6: Screenshot of the Event Editor Context Page

4.2.3 Connection Editor Page

The Connection Editor Page is consists of a full featured graphical editor wrapped in the form of a page. The editor/page can be only used if the event's type is Connection Link Change. This type of change is the most sophisticated because here the developer can also add a compound change and, as its name suggests, it can alter connections and disconnections. It is extended from a standard graphical editor with a palette defined in the Eclipse SWT World. The connector, disconnecter and selection tools can be used from the palette, devices can be added to the pane with the “+” sign on the top menu bar. Contributing devices can be also deleted by selecting them and then using a “right click” context menu or the delete button. The devices frame is highlighted with the same color as in the Device Definition Tool and indicates their roles (Blue-Member, Green-Gateway and Red-Coordinator). If the user hovers on a device a tooltip shows its most important attributes the id, name and role.

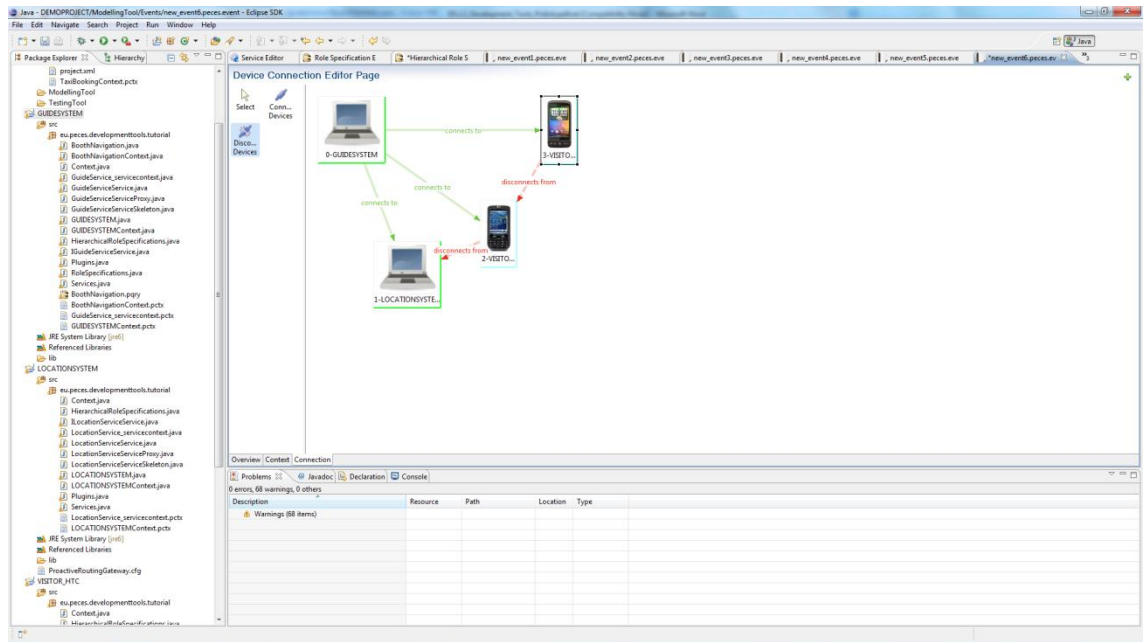


Figure 4.7: Screenshot of the Event Editor Connection Editor Page

4.3 Event Diagram Editor

As I describe above, the Event Editor and the Event Diagram Editor are the Event Manager's two editors for the Dynamics definition. The benefits of the tools were also described. Basically these two editors prepare the model for the Testing Tool which can be imagined as a sophisticated and extended debugging tool for the business logic validation. Event Diagram Editor is important to build the sequence by using discrete event created by Event Editor. This sequence is exported as a XML file and is used by Testing tool to emulate the system.

When the developer has defined the needed events, the sequence of the events can be easily defined with the Event Diagram Editor. During the definition the contributing events can be added immediately but if the developer wants to change the added event set she/he can do it by clicking on the "+" sign at the already opened Event Diagram Editor. A single event can be added multiple times to the sequence. By double-clicking on the event icons the Event Editor will be opened with the corresponding event immediately and the event's parameters can be altered. The export of the events which can be done with the icon next to the "+" icon will generate an events.xml file () with all the information needed by the Testing Tool. Its location is in the file Modelling

Tool/Events/events.xml. Multiple event diagrams can be defined. The Testing Tool always uses the diagram that was exported last to the events.xml file. The event diagram's name from which the events.xml was generated is stored in the events.xml's header. Export can be only done if the editor's state is not "dirty" and there are no separated events in the graph. Events and connection can be deleted from the graph by selecting and using right click context menu or delete button. *The events.xml file generated for this example application is presented in Appendix 4.*

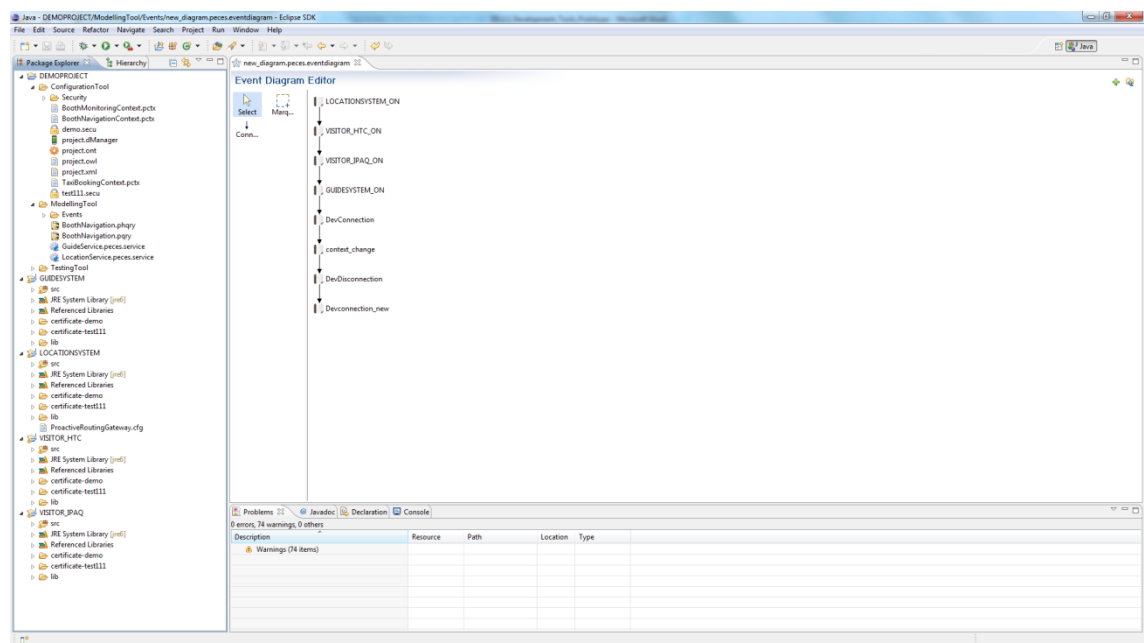


Figure 4.8: Screenshot of the Event diagram Editor

4.4 Summary

In this Chapter, we presented the Modelling Tools which for defining dynamic environment and event. Modelling tool require essential data from Configuration Tool, such as device attribute, device context definition, service mapping and security information (if needed), to do dynamic modelling. The output of modelling tool is written in an event.xml file which contain the sequence of event will be used as an input script to run in Testing Tool.

Chapter 5: Development Tool Design: Testing Tools

This chapter describes the architecture, modular and design of Testing Tool. In Section 5.2, we will discuss architecture include Execution Engine, Connection Engine, Event Engine and Instantiation Engine. We also describe the structure of log file which used as an output of testing environment. This chapter mentioned the middleware modular which is implemented for interaction with middleware component in Testing Tool in Section 5.3. In last content section, Section 5.4 it shows the design and Implementation of Testing Tool and demonstrates the way of Testing Tool working. Normally, we finish the chapter with a summary in Section 5.5.

5.1 Introduction

The main task of the Testing Tool is to support application developers to execute the application defined by the Configuration Tool and the Modelling Tool. The Testing Tool also provides a mechanism to analyse and evaluate the test results produced during the test execution. The Modelling Tool provides a "ready to run" test model to the Testing Tool which is responsible for the execution of the defined information and pass relevant information to the middleware components. As discussed in the previous two sections, the Configuration Tool and the Modelling Tool provide complete deployed entity of the smart spaces, devices and context properties. Testing Tool has to execute the application, record necessary information about the test results in a log file and provides mechanism to evaluate test result to the developers.

The Testing Tool allows application developer to start, stop and suspend the application and provide evaluation of the application results in a preferred format. If the application does not start smoothly, the Testing Tool probably generates some error indications or gives some information about the problems to the application developers. For example, if the application developers defined a device as part of two smart spaces, it provides an error message to the developers that a device can be part of a smart space at the time. In order to test the PECES middleware functionalities in different context, connectivity and security considerations, the Modelling Tool and the Configuration Tool allows the developers to simulate certain values of context changes and connection

changes and then the Testing Tool allows executing the changes and analysing the new results with the Testing Tool.

As discussed in other sections, PECES development tools focus on the novel functionality of the middleware such as role assignment concept, context changes and security issues. These functionalities are very important for application developers. The developers will be interested in testing these functionalities for their applications in many different context and connection settings. For this purpose, the Configuration and the Modelling Tool provide dynamic context and connection information to the Testing Tool. The Testing Tool provides mechanism to parse relevant dynamic information from the Modelling Tool output text file and enables these changes are executed with PECES middleware components.

The Testing Tool also provides a mechanism to generate log files to record the results of the application which is necessary to evaluate the test. To create the log files, the Testing Tool uses the logging functionalities provided by Java API. Log file contains several types of information, such as event descriptor, timestamp of that event and context values, message communication details, etc. This detailed log file is used to analyse and evaluate the result of the middleware functionalities. As discussed previously, the Testing Tool focuses on testing novel feature of the PECES middleware such as role assignment concept, context changes and security features but not the application and log file only needs record any properties related to the novel features of the PECES middleware.

Is it possible to design a tool that will aid the application developer in emulating and testing the smart space system which defined by Configuration Tool and Modelling Tool and increase their productivity? In particular we assume the following questions:

- How to loading the sequence of discrete events as the script for emulation?
- How to execute and control the device instance?
- How to build a connection between devices?
- How to manage all device instances and emulate the discrete event?
- How to get a understandable feedback for evaluation?

For answering the above research questions, the main contributions are listed below:

- Testing Tool Architecture
- Testing Tool Event Engine
- Testing Tool Connection Engine
- Testing Tool Instantiation Engine
- Testing Tool Execution Engine
- Log File generate by testing tool
- Visualization page for showing the emulation process

All these contributions will describe in the following sections.

5.2 Architecture

The Testing Tool can be considered as the union of several parts: the Execution Engine, the Connection Engine, the Instantiation Engine and the Event Engine. The main part is the Execution Engine which is responsible for execution and supervision of the test. Connection Engine, Instantiation Engine and Event Engine cooperate with Execution Engine and are managed by the Execution Engine. Connection Engine will be responsible for the device connection related information and its dynamics. Instantiation Engine is responsible for invoking instantiation command received by the Execution Engine. Event Engine will parse information from the event description file generated by the Modeling Tool. When an event reaches the timestamp, the Event Engine sends information to the Execution Engine and then this information is provided to Instantiation Engine. The Figure 5.1 below shows their interactions, inputs and outputs information.

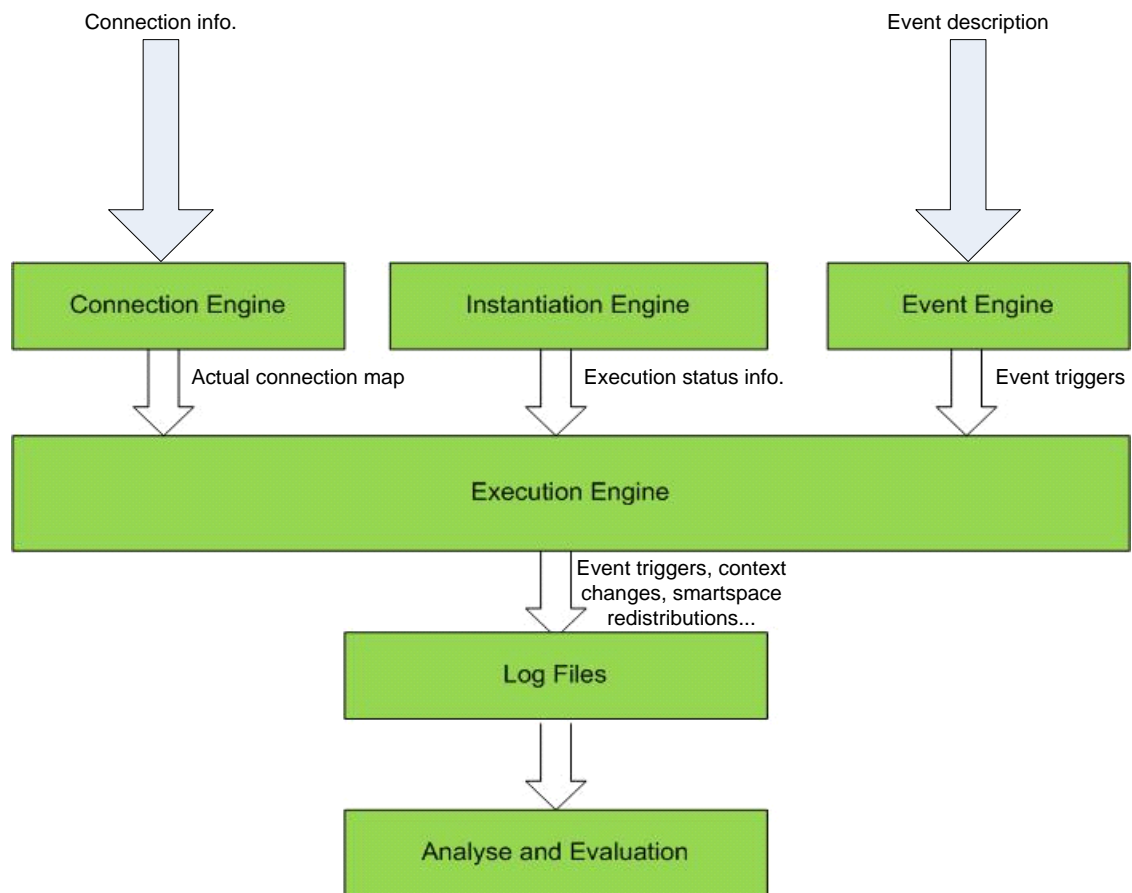


Figure 5.1: Testing Tool Architecture

5.2.1 Execution Engine

The heart of the Testing Tool is the Execution Engine which controls and coordinates all other part of the Testing Tool. The Execution Engine is responsible for the execution of the defined PECES middleware instances by coordinating the information from the Connection Engine, Instantiation Engine and Event Engine. It is also responsible for recording necessary information of the test results to a log file. The Execution Engine supervises all JVM process and is capable of controlling any specific process when a request is received from the Event Engine. This is easily done by the Execution Engine as all JVM processes are running on a single development PC. The `java.lang.Management` package and the `java.lang.Runtime` package APIs are used for management and supervision of the JVM processes.

5.2.2 Connection Engine

The Connection Engine is responsible for providing device connection related information to the Execution Engine such as which devices are connected and which devices are disconnected at a given time. The Modelling Tool output file provides device connection information and the Connection Engine has to parse relevant information from the output file. Due to the dynamic nature in the device connections, the Execution Engine coordinates with the Connection Engine which provides information of changes in device connection. The Connection Engine provides connection mapping information to the Execution Engine by parsing the connection information received by the Modelling Tool. The Connection Engine is responsible for control and supervision of the network topology. As discussed in the Modelling Tool section, BASE transceiver emulator connects instances as found in the connection mapping file using java socket communication mechanism. The BASE transceiver is modified to handle the connect/break request received from the Execution Engine. Java socket programming functionalities (`java.net.Socket` and `java.net.ServerSocket`) can be used for Inter Process Communication (IPC) where objects are transmitted over sockets through the use of `ObjectInputStream` and `ObjectOutputStream` classes. Inter Process Communication (IPC) mechanism allows interaction between instances running in a Java virtual machine in same machine.

5.2.3 Event Engine

This part of the Testing Tool is responsible for parsing the Modelling Tool event description file and provides information for the Execution Engine. The Modelling Tool uses the timestamp to describe relevant events. The timestamps are defined by the modelling tool has no units and the Testing Tool has assign a scale for the timestamps. It would be possible to assign both fixed time scale and different time scale along with the simulation time. Events are ordered according to the timestamps and once a timestamp reaches the event that event will be fired. The event types provided by the Modelling Tool are the device switch on and off information, context change and link communication creation and link lost.

The Event Engine is responsible for time control. When the test starts, the Event Engine performs time control. When the event reaches the timestamp,

the Event Engine provides necessary information to the Execution Engine as defined in an event description file which is generated by the Modelling Tool. It is very important that all events generated by the Modelling Tool get processed by the Testing Tool during the execution phase. The Testing Tool records all processed events in the log file where any failed events can be identified later during the evaluation by the application developers.

The Modelling Tool section discussed about the ability of scaling the timeline of the events. This feature enables the developer to fire quickly (relative to real time) less interesting events and then eventually raises the scale at more interesting parts to be able to follow the changes in real time. For this purpose the Event Engine has an alternative time measurement mechanism where this event timescale is measured relative to the scale set up by the developer. For example, if there is a scenario where the developer sets 3 events which fire at the 2nd, 5th and 12th second and when the scale is raised to 2 from the normal scale (assume 1), the events will be fired relatively to real time after the 1st, 2.5th and 6th second.

5.2.4 Instantiation Engine

This part of the Testing Tool is responsible for instantiation of the PECES entities. The input for the Instantiation Engine is given by the Modelling Tool output device description files. The Modelling Tool provides a complete virtual device configuration where Instantiation Engine will run as a JVM process. In the testing process, the Event Engine cooperates with Instantiation Engine through the Execution Engine. When an event defined at the event description file reaches the timestamp, the Execution Engine informs to the Instantiation Engine to start the corresponding process. The Execution Engine monitors all process and provides information to the Instantiation Engine when to stop or start the specific process. The Instantiation Engine has to map which processes belong to which devices. An easy and elegant way of doing this is to name each process with the device name it represents. There are a few wrapper utilities which can be used for this purpose. Normally when a JVM process is started, the running process list will be shown as java.exe or javaw.exe. The platform independent utility such as launch4j can be used for this purpose to name the processes. This wrapper names the process after instantiation the virtual

devices and they will be listed in the OS process list by their names (device1, device2, device3). This makes easy for the Instantiation Engine to manage them by referencing their name and no further mapping file is needed where the PID name mapping has to be defined. When the Execution Engine receives information about the device to be switched off from the Event Engine, it provides this information to the Instantiation Engine which is responsible for completely shutting down corresponding JVM. When the Instantiation Engine receives instruction to be switched on, it will start the JVM defined by the Modelling Tool. The `java.lang.Runtime` package APIs can be used to shutdown any specific JVM process.

5.2.5 Log Files

The application developer may want to have detailed access to all interesting information of the PECES middleware performance related parameters. During the execution, a log file is generated with timestamps and other related information. Log file records information such as:

- Event occurrences
- Context values
- Smart space member devices
- Changes on context values
- Change on smart space member devices
- Received/sent messages
- Event description
- Device ID

The log file can be analysed by the application developers by defining constraints with timestamps. The validation of the constraints can be shown when the simulation reaches to the specified timestamp. Constraints can be defined with context values and this value will be checked with value in the log file. Multiple constraints can also be defined in this way. Application developers may define the expected number of devices in any smart space (e.g, 10) for a given time (e.g, 100 seconds) and validate these constraints from the log file results. The developers can validate the context changes effect by checking in the log file whether expected results are produced. If the log file has well detailed information about the test results, application developer can use this

data to get comprehensible information about the test. Recording all detailed information into the log file may generate huge file which may cause some problem. Due to this, log file only record the relevant information to be used by the developers. The Execution Engine must have an interface to be informed after an event convergence delay is expired and to gather all the needed information for the other modules and dump it to the log file. This can be imagined as a scenario where the Event Engine has the information about the convergence delay of an event. When the delay expires the Event Engine makes a query to gain all the interesting attributes from the virtual devices and/or the Connection Engine. When the query is done the information is sent to the Execution Engine via the mentioned interface and it will be dumped to the log file. For the query process the modules and the devices must have an interface, which consist on an IPC socket, with a service that serves predefined queries about the needed attributes.

5.2.6 Analysis and Evaluation

From the log file, application developers are able to examine relevant features of the middleware which are important to their application. Log file will only be available once the test process is completed. The developers can analyse middleware test results and easily map the meaning of the results. Application developers can easily modify the scenarios using the Configuration and the Modelling Tool can generate new test model and produce new test results with the Testing Tool. The Testing Tool allows comparing the old scenario results and with new scenario results. A Diff tool can be used to analyse two different log files with different scenarios where application developers can easily visualise the difference in the results. Also Eclipse provides open source log and trace analyzer tools (e.g., The Eclipse Test and Performance Tools Platform (TPTP)) which can be used to analyse the log file results. These features in the Testing Tool help application developers to check and study how the smart spaces and the devices perform as changes occur. For example, application developers are able to find out how the smart space formation are evolving and how many devices are taking part in a given time, etc..

The current version of Testing Tool does not have separate mechanism to support fault Injection, but the testing tool is stable enough when some faults

occurred. As describe in modelling tool, a sequence of discrete events will be generate by Event Manager. During the process of generating the sequence of events, some faults may exist in the sequence. These faults may make in purpose or not. In generally, the potential faults are listed below:

- Switch on a device already switched on
- Switch off a device already switched off
- connect two devices which already connected
- disconnect two devices which not connected
- connection a switched off device to another one
- context properties change in a switch off device

The first four fault types won't cause any change in testing too, the testing tool automatic ignores these error events and send a message in log file to let developer know. The last two fault types will cause the exception. This exception will be thrown and get by log server. The error message will appear in log file with all necessary information, such as, time, device ID, etc..

5.3 Modules

PECES Testing Tool has many modules and components to perform test and record the results of the test performed. As discussed in the previous sections, the Testing Tool has several parts which are responsible for execution, connection, event and instantiation and logging. Testing Tool modules can be defined as two main modules:

- Central Control Module
- Middleware Modules

Central Control Module is the main module which is responsible for the functionalities of the Execution Engine, the Connection Engine, the Instantiation Engine and the Event Engine. The Middleware Module is implemented as an extension to PECES middleware component and it integrates the Central Control Module with the PECES middleware components.

The Central Control Module handles most of the Testing Tool tasks such as scenario control, device switch on and switch off and implements internet level registry. The Middleware module has three sub-modules: networking, context

and logging. The following sections will explain briefly about the Central Control module and the Middleware modules. The Figure 5.2 shows high level architecture of the proposed modules.

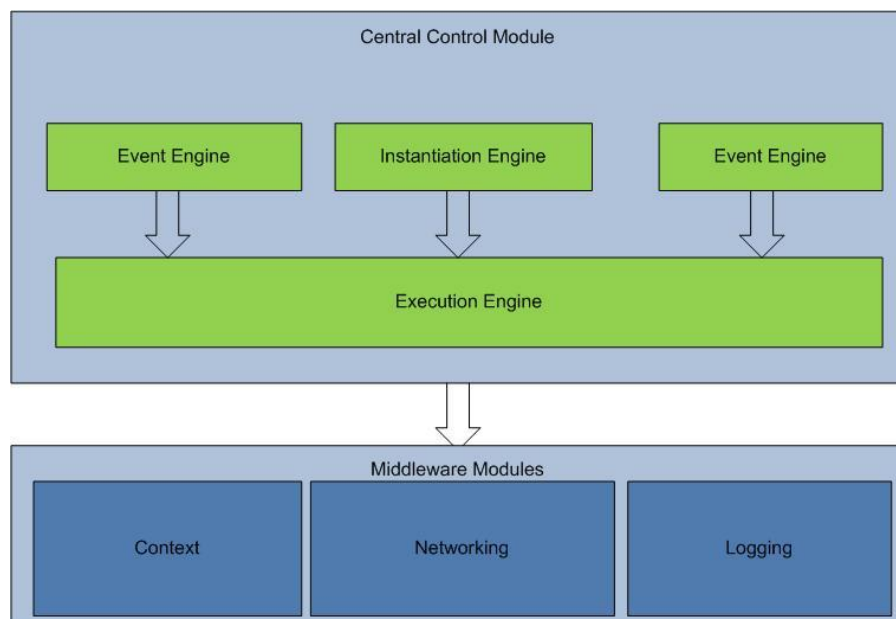


Figure 5.2: Testing Tool Modules

5.3.1 Central Control Module

5.3.1.1 Overview

The Central Control Module will be responsible for test execution and acts like a supervisor of the simulation as a whole. It is the underlying layer for the connection, instantiation and event engine providing the interoperability between them. Basically, this module implements the Execution Engine, the Event Engine, the Instantiation Engine and the Connection Engine functionalities and other necessary interfaces. This module is also responsible for providing the log information of the test execution, as well as an interface for other processes to attach themselves during the execution. The context information and other connection related information defined by the Modelling Tool has to be injected into the middleware by the Testing Tool. The Central Control Module provides parsers which are used for translating the Modelling Tool output such as connection information and event description into the required format.

5.3.1.2 Central control over the scenario

In the development process, the Configuration Tool provides initial static information to the Modelling Tool. The Modelling Tool uses this initial information and adds additional information related to smart space connections, events, etc. The outputs of the Modelling Tool provide connection information, event definitions, etc which is used by the Testing Tool to execute required testing scenarios defined by application developers. This module is responsible to parse and manage the information from the Modelling Tool output files. Output files interpretation of the necessary data will be passed to the middleware component to run the scenarios. The Central Control Module also cooperates with middleware logging module to provide test results output of some important parameters such connection changes, context values and smart space related information to the log file.

5.3.1.3 Starting and stopping devices

Device switching on and switching off adds dynamics to the test. The Central Control Module supervises this process and provide mechanisms to switching on or shutting down the device JVM as information provided by the Modelling Tool. The Modelling Tool provides a complete virtual device configuration that has to be run on a JVM by the Testing Tool. The Modelling Tool provides the timestamp and virtual device ID to the Execution Engine. The timestamp provides information about when switch on and switch off event will happen and the virtual device ID identifies the JVM to stop or start. For this purpose, the Testing Tool has to manage and record all JVM process by mapping them to their corresponding virtual device. Switching on means starting the JVM with the full environment defined by the Modelling Tool, while switching off will mean the shutdown of the corresponding JVM. This module provides mechanism to effectively create and manage the virtual devices for this task.

5.3.1.4 Internet level registry

The `eu.peces.communication.registry.internet` package is responsible for providing discovery for the services and roles across smart spaces. The devices in the smart space can use coordinator and gateway in the smart space to access and publish information on the internet level registry. The service

information that is available includes information about the service name, service description, service context information and the address of the target device, coordinator and gateway with necessary plug-in description. Space-level registry acts as a liaison to export, un-export and search the internet-level registry and provides necessary forwarding mechanism between the device and internet-level registry using the gateway role. As already discussed in the Modelling Tool section, an additional “meta-node” representing PECES compatible Internet registry which is capable of connecting with the device level component and space level component provides necessary forwarding mechanism to allow interaction between the applications and internet-level registry during the development. The “meta-node” is initially configured by the Configuration Tool and then additional information will be added by the Modelling Tool. Having a “meta-node” concept to model internet registry will fulfil the development process without breaking any PECES middleware functionality. This Central Control module is responsible for implementing the meta-node internet level registry concept in the development environment.

5.3.2 Middleware Modules

5.3.2.1 Overview

Middleware Modules is implemented to interact with PECES middleware components. The Central Control Module is responsible for supervision of the test process and will have information about the necessary connection changes, context changes and other related information. This information will have to be properly injected to the middleware. The Middleware Modules integrates the Central Control Module with PECES middleware to achieve this objective.

5.3.2.2 Networking Module

The connection information generated by the Modelling Tool is used by the Testing Tool. The Networking Module uses the connection information from the Modelling Tool output file via Central Control Module. The connection scenario may define several devices which are connected in different time. The Networking Module interacts with middleware component such as coordinator, gateway and device functionalities to inject the information received from the

Execution Engine. Actually, this module interacts with the Connection Engine through the Execution Engine.

The Networking Module is responsible for handling device networking related information such as which devices are connected and which devices are disconnected during a period of time. This module is implemented as an extension to the PECES middleware networking related component with the only purpose to manage and simulate the network connections. During the execution phase, a proper supervision of the network connection is necessary due to the dynamics in the device connections. Connection dynamics is handled by this module with the coordination of the Event Engine through the Execution Engine.

5.3.3.3 Context Module

The `eu.peces.middleware.context` package contains the context provisioning component that provides access to the context information. This context information is necessary to perform the role assignment. The context provisioning component allows the modelling of the context information according to the PECES Context ontology and Query specification. The main task of the context provisioning component is to support the storage and retrieval of the context information using queries. This context module enables the access to the context information defined the application developers at the Modelling Tool and the Configuration Tool. As discussed in the Modelling Tool section, context dynamics can be handled using stateful concept or stateless concept. The PECES development tools implements the stateless concept, since it is considered satisfactory to handle context dynamics. This module is responsible for providing context information to the middleware by cooperating with the Central Control module.

5.3.3.4 Logging Module

As discussed earlier, the Testing Tool records test results to a distinguished text output called the log file. Log file records several parameters such as event description, timestamp, context values, smart space devices, smart space name, received/sent messages and any other information which are necessary to

evaluate any specific application scenario. This module is responsible for record necessary information of the test results.

The Logging module is responsible for producing and managing log file which will be analysed by the Testing Tool. This Logging module makes sure that logging can be as inexpensive as possible and record necessary information to one log file. Logging all test related information to the log file may slow the application affecting PECES middleware performance, hence the test results.

The available logging API in the java package `java.util.logging` is used to log necessary information such as role changes, context changes, communication data and access control related information. The Java logging API is part of J2SE of JDK 1.4 and greater, and it ships with the JDK. It is designed to let a Java program to record messages of interest to the application developers. All log messages can be sent to a TCP port where this module is listening and registering the output of all devices. This is already implemented in the JAVA package.

5.4 Testing Tool Implementation

5.4.1 Execute page

The initial device status information is displayed (for example, all devices are “OFF”) in the Testing Tool multi-page editor Execute Page. Application developers are able to define the required time to test the application specified by the previous tools. Developers can also provide Internet Registry IP and port information if they want to test that application with the internet registry. The Figure 5.3 shows the defined application device status before the test was executed.

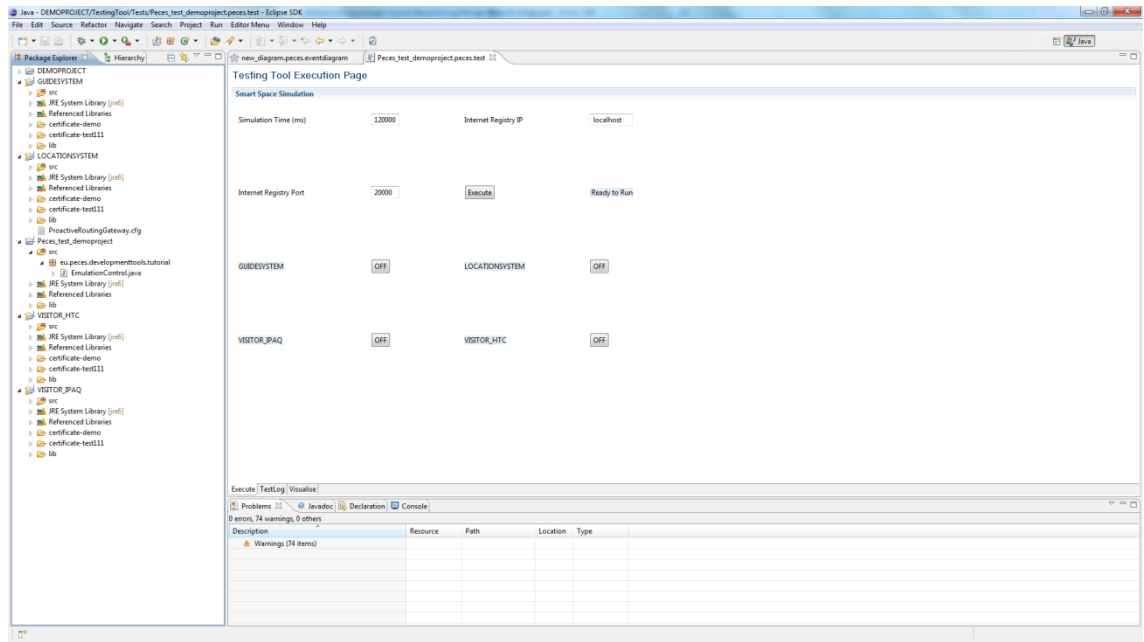


Figure 5.3: Screenshot of the PECES Testing Tool Execute Page

The “Execute” button enables developers to run the application. As seen in Figure 5.4 below, the status of each device is shown during test (three devices are “ON” and one device is “OFF” at a particular time). All middleware and application related information is logged to a single log file with the specific device and absolute time of the system. Using this absolute time, the relevant time between the devices (based on test start time) are calculated for further analysis. The Figure 5.4 shows the defined application device status while the test is running.

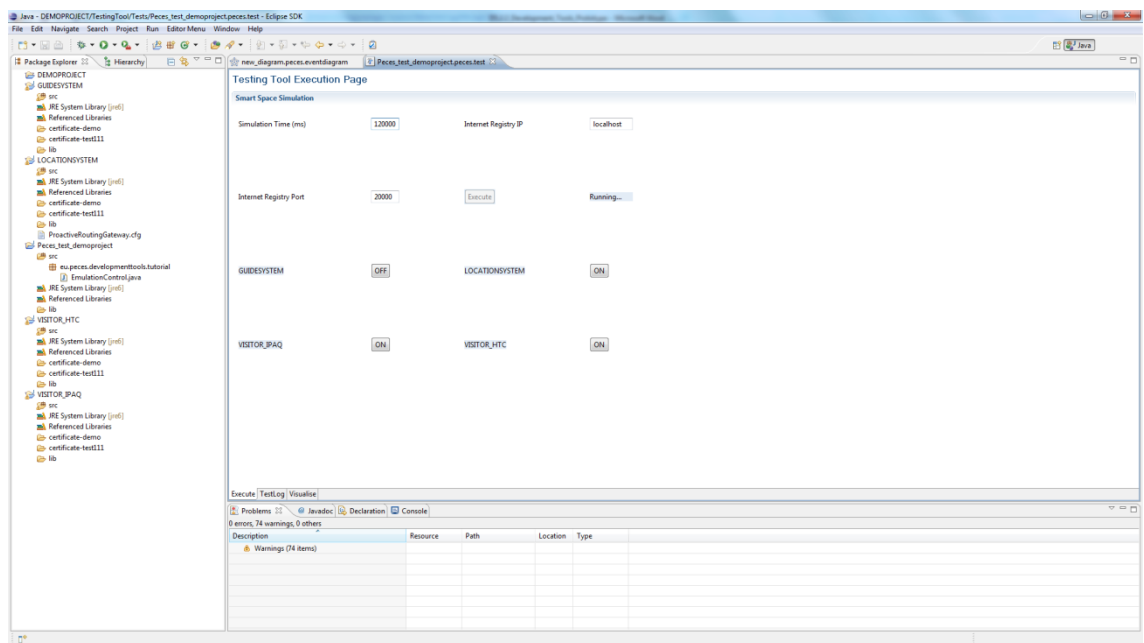


Figure 5.4: Screenshot of the PECES Testing Tool Execute Page during the test

5.4.2 Testlog page

The TestLog Page provides detailed information of the test performed. Middleware and application related events are logged with specific device name, absolute time and other relevant and available information. Figure 5.5 shows the screenshot of the Testlog page.

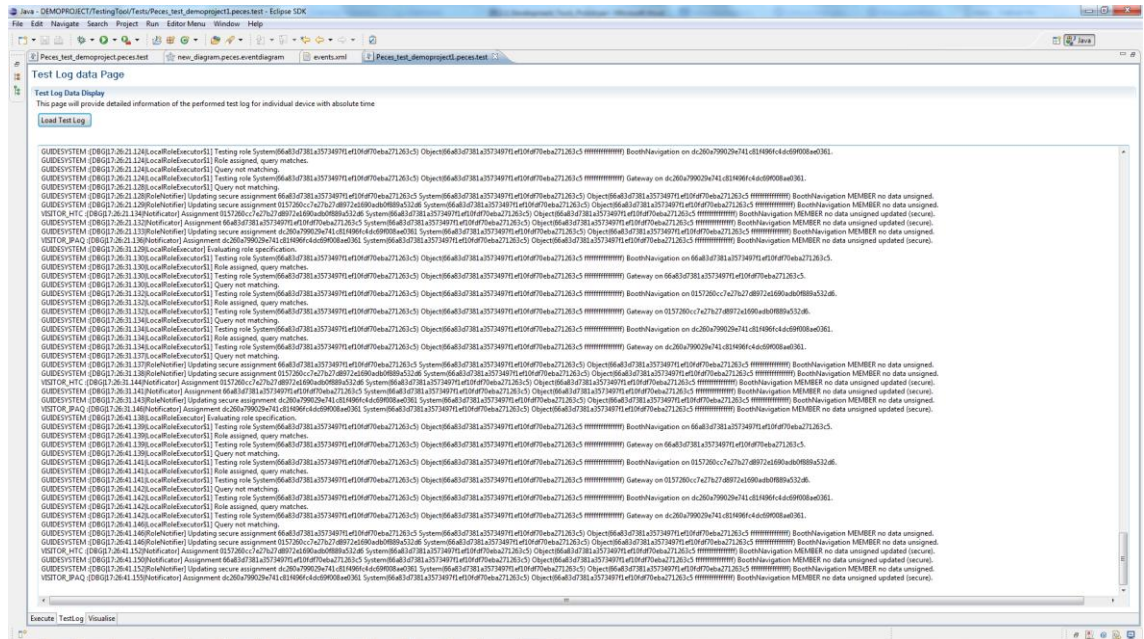


Figure 5.5: Testing Tool Testlog page

There are two sources to generate the information in log file. The first one is device instance. Information from single device shows the status of this device at certain time when an important event happened, such as switch on, join smart space, etc.. The other source is EmulationControl which is the global engine to aspect the whole system. Logging from EmulationControl describe the global events, relations between devices and faults such as smart space established, adding connection between devices, etc.. Log file record all important information when emulator running. Log file can be analyzed to help developer to inspect the system running situation. Developers can check whether the system running as they expect. If some errors during the emulation, log file also helps the developer to debug and improve the design of the smart space.

Here are some event logs of the defined example application in this document:

```
GUIDESYSTEM :[DBG|17:25:00.952|Notificator] ..... BoothNavigation MEMBER no data unsigned assigned (secure).
```

The above log provides information that GUIDESYSTEM is assigned as BoothNavigation smart space at absolute time 17:25:00.952.

EmulationControl :[LOG|17:25:10.906|EmulationControl] Adding connection between 66a83d7381a3573497f1ef10fdf70eba271263c5 0157260cc7e27b27d8972e1690adb0f889a532d6

The above log provides information about adding a connection between the devices GUIDESYSTEM (its system ID is 66a83d7381a3573497f1ef10fdf70eba271263c5) and VISITOR_HTC (its system ID is 0157260cc7e27b27d8972e1690adb0f889a532d6) at absolute time 17:25:10.906.

EmulationControl :[LOG|17:25:25.918|EmulationControl] Removing Triplet URI(<http://www.ict-peces.eu/ont/device.owl#LOCATIONSYSTEM>) URI(<http://www.daml.org/services/owl-s/1.1/Service.owl#provides>) URI(<http://www.daml.org/services/owl-s/1.1/Service.owl#LocationService>)

This above log provides information of a triplet removed (the LocationService) from the LOCATIONSYSTEM at absolute time 17:25:25.918.

5.4.3 Visualisation page

The Testing Tool Visualise Page provides features to visualise the smart space network status based on the test log data events with relative time. This Visualise Page lists analyzed important events occurred during the test. The “List of Events” may contain event such Device Switch ON, Device Switch OFF, Connection, Disconnection and Smart Space Establish, Smart Space Join, etc. The status of the system can be viewed by double clicking on the name of the specific event. For example, by double clicking on the fourth event (Device 0 on) in the list, Figure 5.6 below shows the visualisation of the system at time 11991 ms (after test started). It displays the four devices which are available (already switched ON) at that time and the devices are expected to form a smart space defined by the Role Specification Tool. This also shows that devices are available but there is no communication between them.

Double clicking on the particular event from the “List of Events” provides not only the particular event but also the status of the whole network and its devices, its connections and smart space activities, etc. at a particular time.

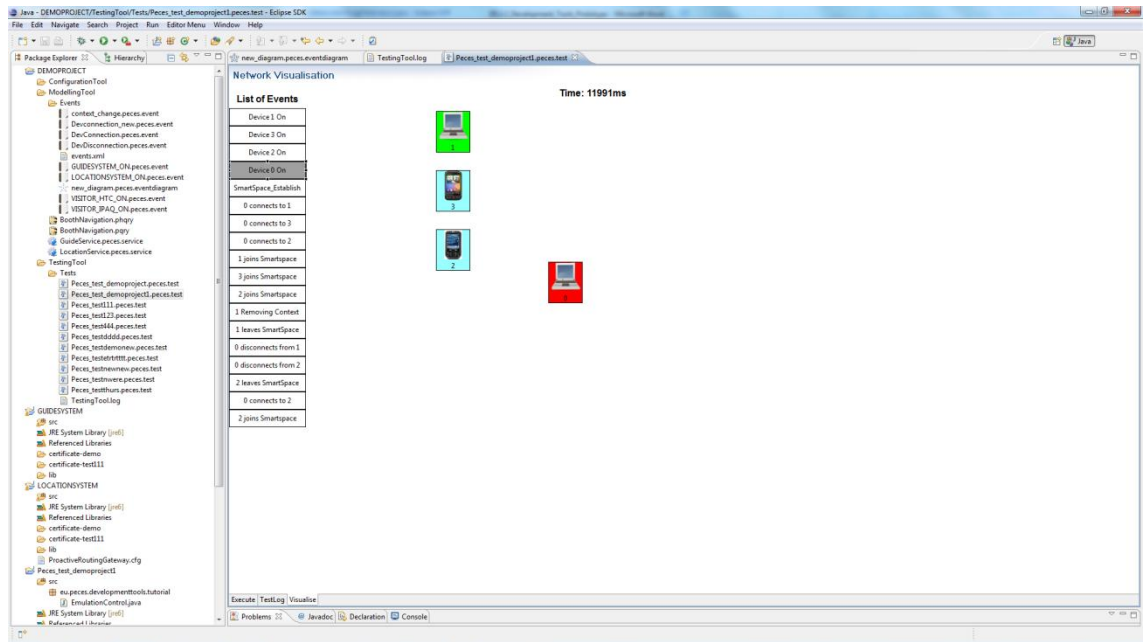


Figure 5.6: Screenshot of the Visualise Page just after four devices were switched ON

The Figure 5.7 shows that coordinator role as “BoothNavigation” is assigned and the coordinator is looking for the devices based on the context information to form smart space. Still there is no communication between the devices (no connections are seen between the devices).

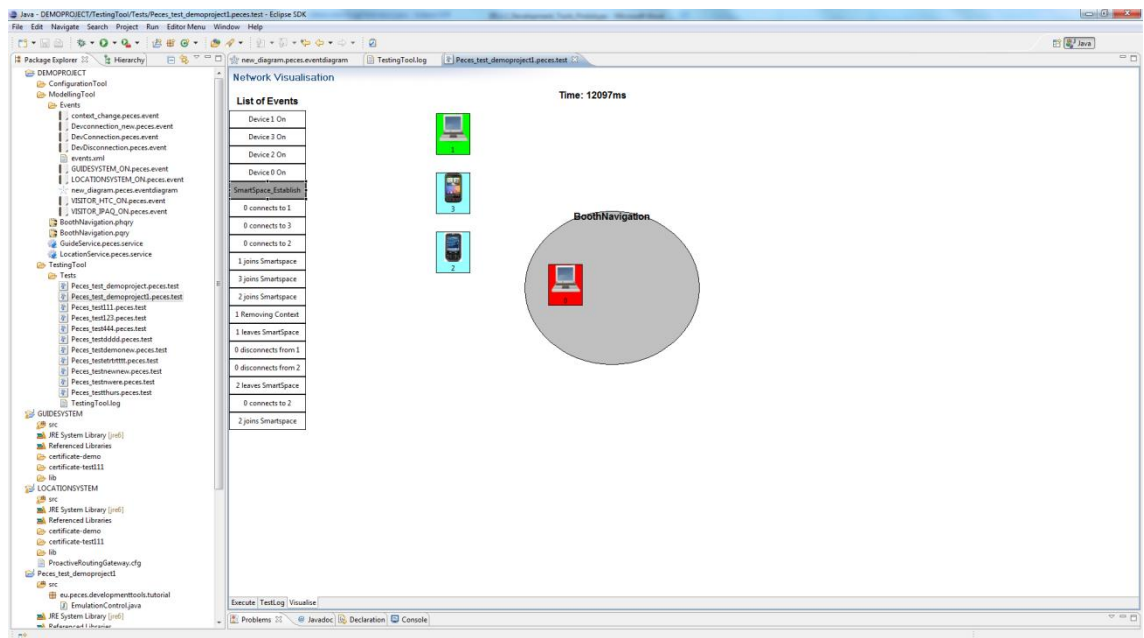


Figure 5.7: Screenshot of the Visualise Page just after Smart Space established only with GUIDESYSTEM

The Figure 5.8 shows that the three devices were connected with the coordinator device which is assigned with the “BoothNavigation” role and devices are expected to join in the smart space based on their context information.

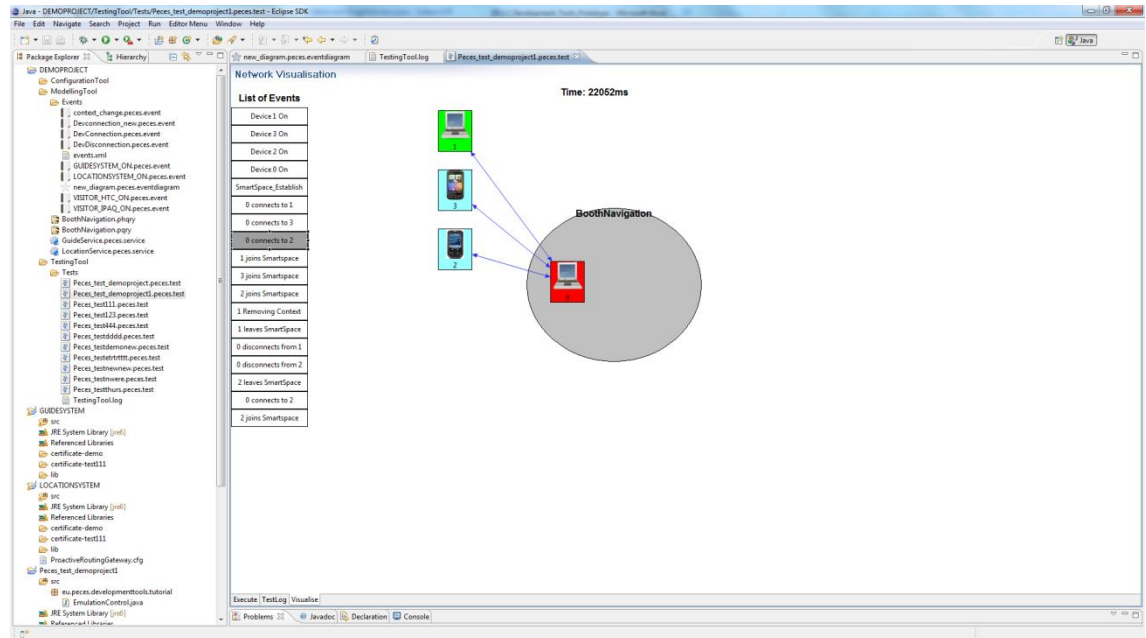


Figure 5.8: Screenshot of the Visualise Page just after devices were connected with the GUIDESYSTEM

The Figure 5.9 shows that based on the context information and role specification, the *VISITOR_IPAQ* the *VISITOR_HTC* and the *LOCATIONSYSTEM* joined the smart space with the coordinator.

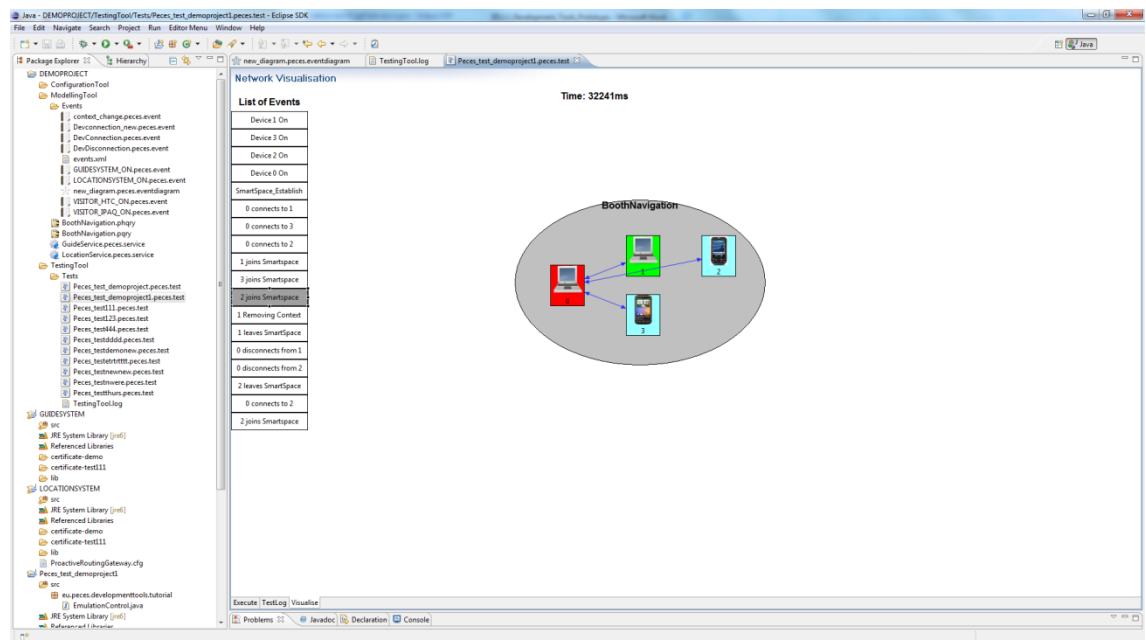


Figure 5.9: Screenshot of the Visualise Page just after devices joined the smart space

The Figure 5.10 shows that the LOCATIONSYSTEM context property “LocationService” was removed from the device and the LOCATIONSYSTEM is expected to leave the smart space when next role assignment takes place.

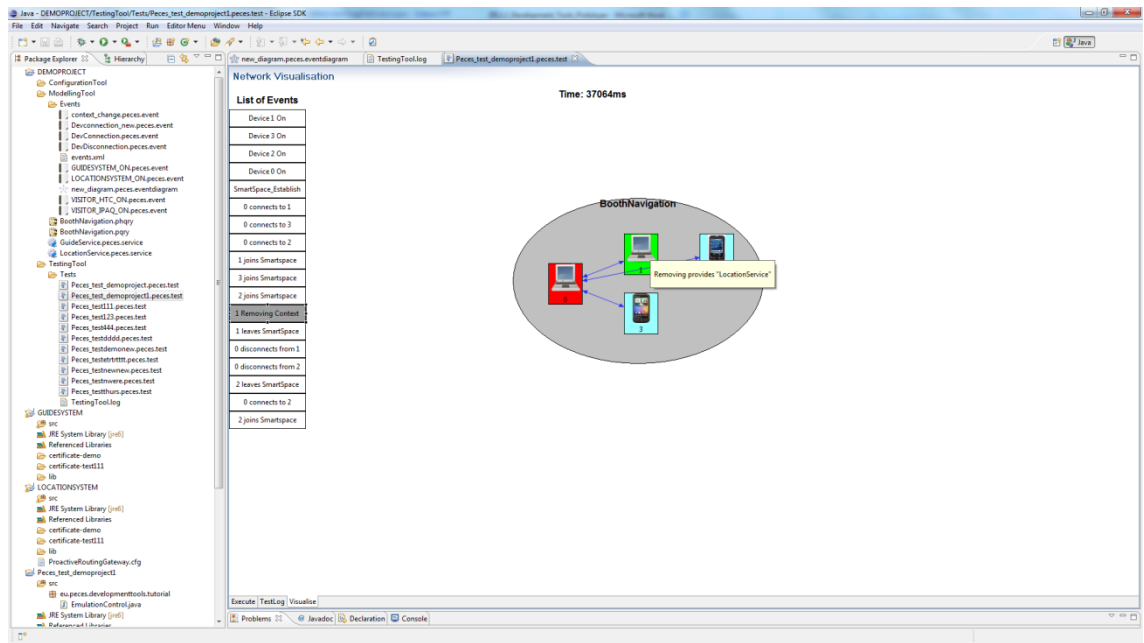


Figure 5.10: Screenshot of the Visualise Page just after the LOCATIONSYSTEM “LocationService” Context was removed

The Figure 5.11 shows that the LOCATIONSYSTEM left the smart space due to its context property changes (“LocationService” was successfully removed).

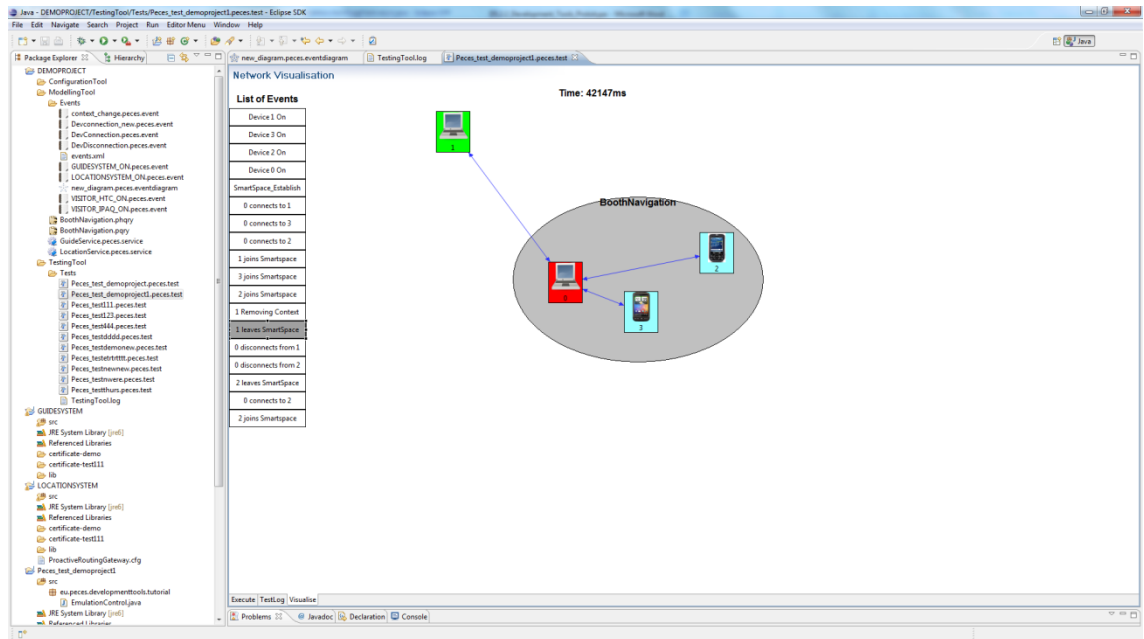


Figure 5.11: Screenshot of the Visualise Page just after LOCATIONSYSTEM left the smart space due to its context change

The Figure 5.12 shows that connection between the VISITOR_IPAQ and the GUIDESYSTEM was removed and the VISITOR_IPAQ is expected to leave the smart space when next role assignment takes place.

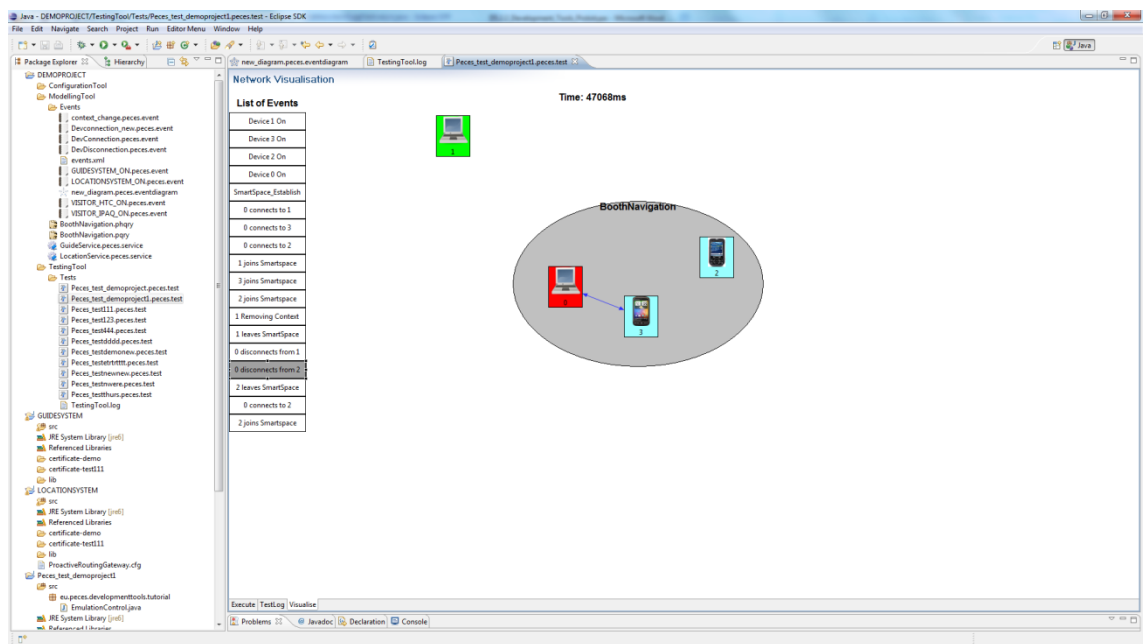


Figure 5.12: Screenshot of the Visualise Page just after VISITOR_IPAQ disconnected from the GUIDESYSTEM

It can be seen in the Figure 5.13 that VISITOR_IPAQ left the smart space because the connection between the GUIDESYSTEM and the VISITOR_IPAQ was removed earlier.

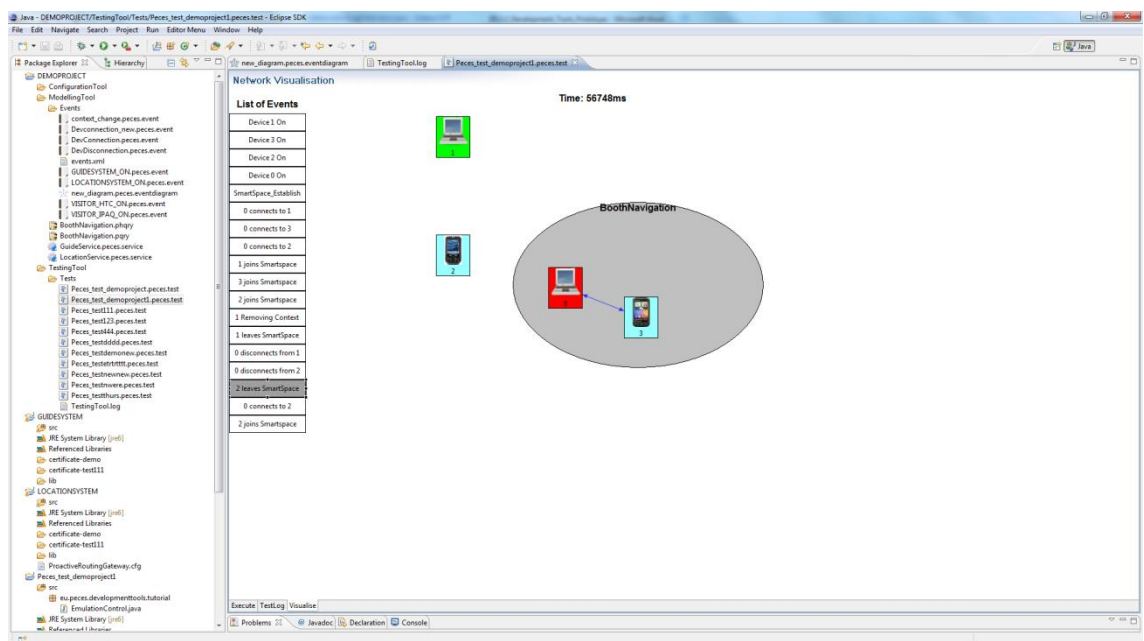


Figure 5.13: Screenshot of the Visualise Page just after VISITOR_IPAQ left the smart space

The Figure 5.14 shows that connection between the VISITOR_IPAQ and the GUIDESYSTEM was established again and VISITOR_IPAQ is expected to join the “BoothNavigation” smart space when next role assignment takes place.

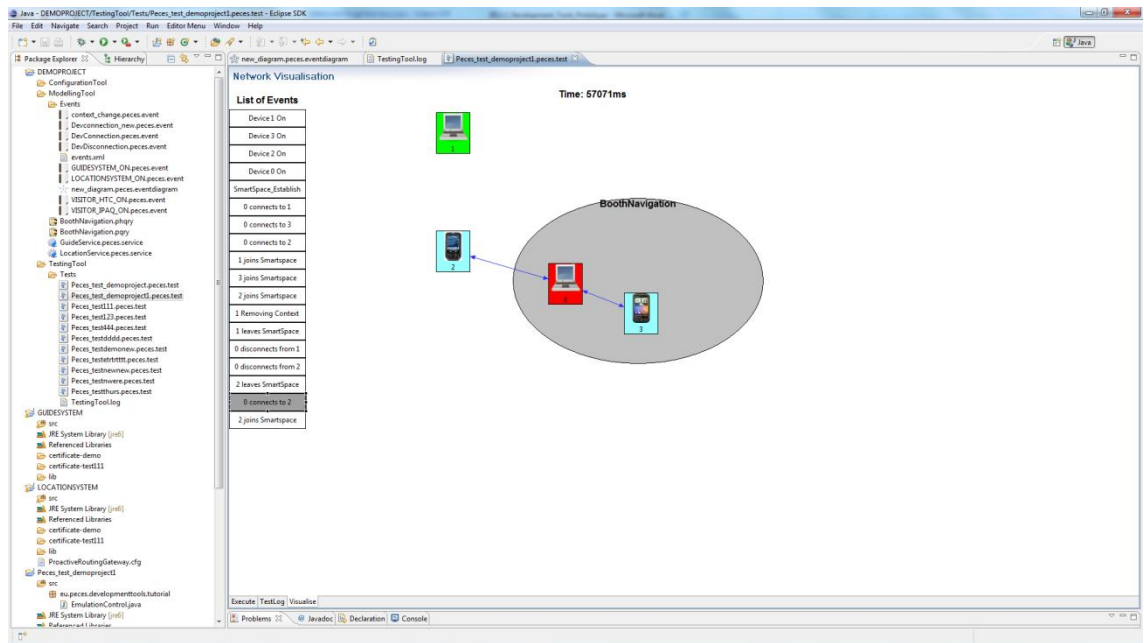


Figure 5.14: Screenshot of the Visualise Page just after VISITOR_IPAQ connected to the GUIDESYSTEM

It can be seen in the Figure 5.15 that VISITOR_IPAQ re-joined the smart space as the connection between the GUIDESYSTEM and the VISITOR_IPAQ was successfully re-established.

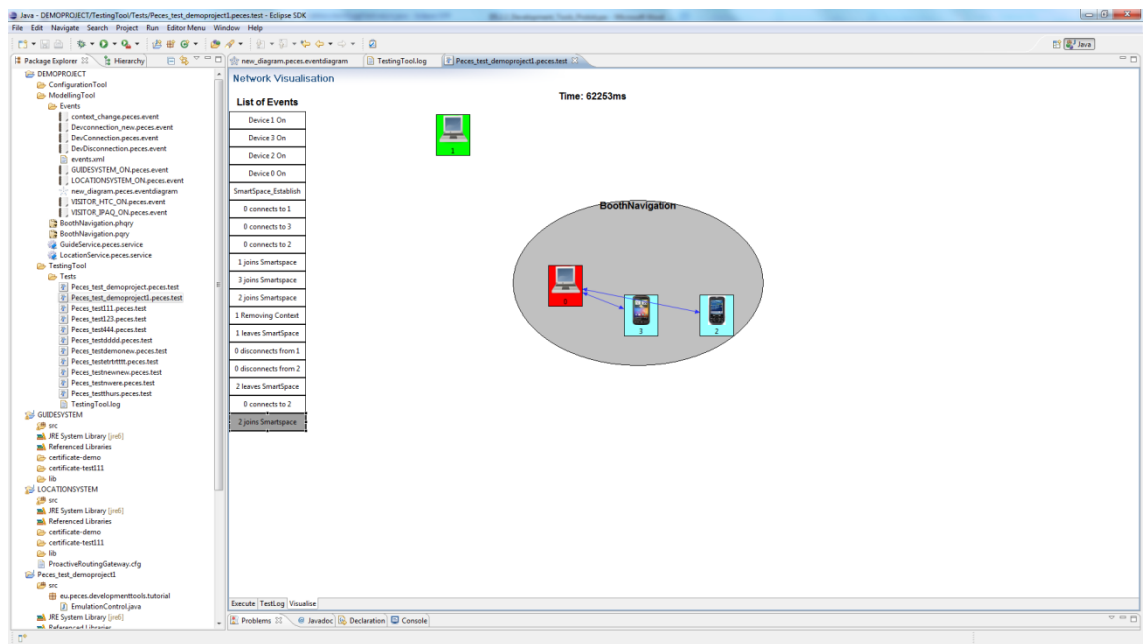


Figure 5.15: Screenshot of the Visualise Page just after VISITOR_IPAQ joined the smart space

Testing tool also support multi-smartspace emulation and the result can be shown in visualization page. For example, by double clicking on the fourth event (TaxiBooking Establish) in the list in Figure 5.16 shows the visualisation of the smart space system at time 3050 ms (after test started). Figure 5.16 displays the two different smart spaces (BoothNavigation and TaxiBooking) with its coordinator devices and it also displays the PECES Internet Registry availability but not connected with the smart spaces at this time event. The smart spaces are expected to form a hierarchical smart space defined by the Peces Role Specification tool.

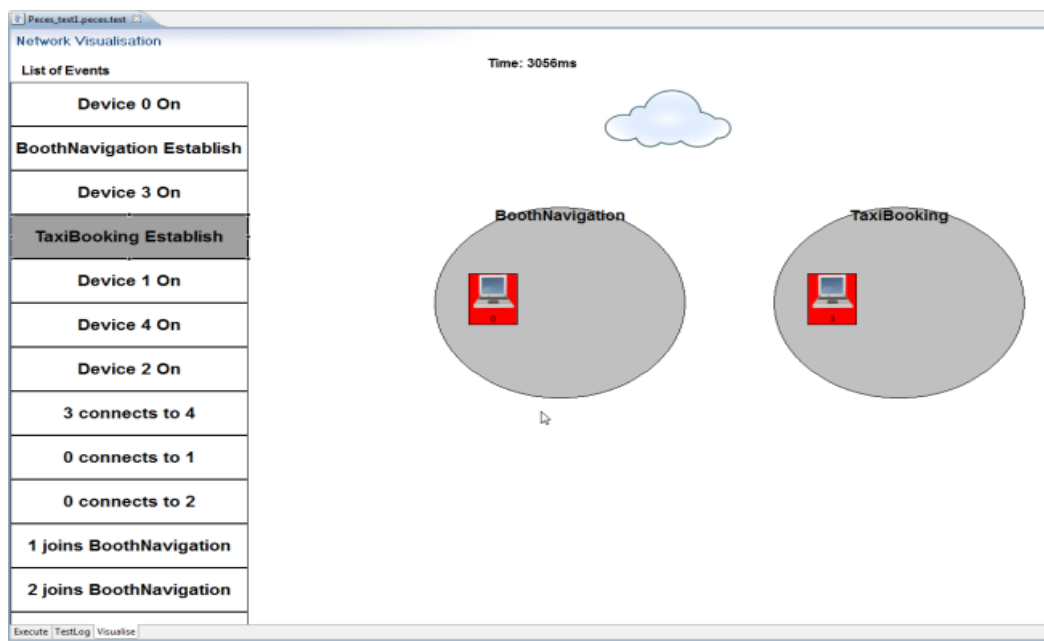


Figure 5.16: Screenshot of the Testing Tool Visualisation with two smart spaces

Figure 5.17 shows that BoothNavigation smart sapce and TaxiBooking smart space are formed a hierarchical smart space using the PECES Internet Registry when it is necessary. This clearly visualise the PECES middleware enables communication between devices in and across different smart spaces in a context dependent and secure manner.

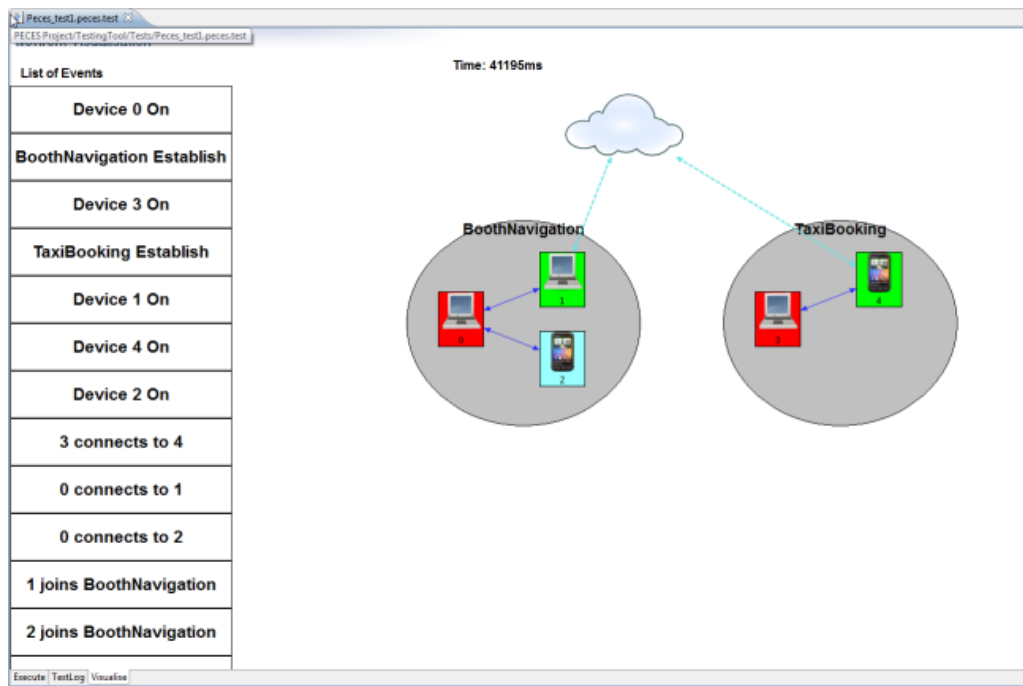


Figure 5.17: Screenshot of the Testing Tool Visualisation with a hierarchical smart space

5.5 Summary

This chapter present the Architecture by describe four different engines:

- Execution Engine: the core engine which control and coordinates all others parts of the Testing tool.
- Connection Engine: provide device connection information to Execution Engine
- Event Engine: get input from modelling tool and provides information which get from the input data to Execution Engine.
- Instantiation Engine: is responsible for instantiation of PECES entities.
- Logging: record all execution information and output a log file for future evaluation.

Two main modules which responsible these engines:

- Central Control Module: responsible for test execution and supervise the simulation.
- Middleware Modules: is used to interact with PECES middleware components.

We also explained the design and implantation of three pages of Testing Tool, Execute page, Testlog page and Visualisation page, and demonstrate how it works and result.

Chapter 6: Use Case Study: build a real Application by using Development Tool

Chapter 3, 4 and 5 present the full set of Development Tools and describe how they help developer to build an application. This chapter demonstrate a real application which using PECES middleware. We will separately describe three specific sub scenarios of the trade show guide system:

- Smart booth navigation: provides automatic guidance to the visitors based on the booth availability in Section 6.2.
- Smart booth monitoring: provides sensor information (temperature and light level) to a remote emergency system to manage emergency situations in Section 6.3.
- Smart taxi booking: allows a visitor to request a taxi without knowing the visitor's whereabouts in detail in Section 6.4.

In each scenario, we will describe the prototype, architecture, design and how to use the development tools to development the application.

6.1 Introduction

The Trade Show Guide system use cases use wireless sensor networks as a platform to provide guidance and navigation information to the visitors at the trade show. For example, the trade show may have many booths and each booth may target a specific technical area. At a particular time, some booths are very busy with visitors and other booths are not. Gathering the availability of each booth at a particular time would be interesting information which can be used to make an intelligent guidance for the trade show visitors. The availability of any booth can be determined by measuring noise level at a booth using microphone. In addition to the noise level, the visitors (with Smartphones) location information can also be used to determine the availability of each booth, as visitors location can be determined with considerable accuracy with WiFi fingerprinting indoor localization techniques. Based on the microphone sensor reading and available visitor's location information, the trade show guide system can provide guidance to the visitors based on the booth availability.

Also temporarily installed infrastructure at any trade shows could be monitored using wireless sensor network for the safety of the visitors. Each booth surrounding temperature and light level reading can be monitored by the wireless devices installed in each booth attached with temperature and light sensors. These sensor readings can be used to provide accurate location information (GPS location and local location) to a remote emergency system if there is any emergency situation detected (e.g. high temperature or light level reading). These warning can also be sent to the visitors at the trade show. By knowing the exact local location of the incident and their current location, visitors can be able to get emergency exit navigation information.

In addition to provide a seamless experience to the visitors at the trade show, this use cases also provide a taxi booking service. This location based booking service allows visitors to request a taxi without knowing his/her whereabouts in details. If the visitors don't have access to GPS location, they can get a GPS location from the trade show guide system. Also the visitors do not have to wait at the same place until the taxi arrives and the visitors can be able to move to another nearby location where the taxi service is able to update the visitors' location changes to the taxi which enable to the taxi to pick up the visitor from a new location.

This chapter describes three specific scenarios of the trade show guide system:

- Smart booth navigation: provides automatic guidance to the visitors based on the booth availability.
- Smart booth monitoring: provides sensor information (temperature and light level) to a remote emergency system to manage emergency situations.
- Smart taxi booking: allows a visitor to request a taxi without knowing the visitor's whereabouts in detail.

In each scenario, we will describe the prototype, architecture, design and how to use the development tools to development the application. The following sections will describe each one of the applications.

6.2 Smart booth navigation

The first use case in the trade show guide system is the Smart Booth Navigation. This use case consists of two visitors (Visitor1 and Visitor2) and a GuideSystem. The GuideSystem is providing the services to the visitors and is also gathering each booth sensor reading and visitors' location information periodically. Using this sensor reading and the visitors' location information, the GuideSystem can be able to provide intelligent guidance and navigation services to the visitors.

6.2.1 Scenario

6.2.1.1 Operation

A PECES application is installed with coordinator and gateway functionalities in the GuideSystem (For this use case, the gateway functionality is not necessary but it is required for the Smart Booth Monitoring use case as the same application will be used).

When the application is started, the GuideSystem establishes BoothNavigation smart space as defined by the role specification. When the visitors (Visitor1 and Visitor2) enter to the WiFi coverage area of the GuideSystem, the visitors expect to join with the BoothNavigation smart space.

The GuideSystem is also installed with Ekahau location tracking software which records visitors' location for every 5 seconds. The GuideSystem is connected to a WSN gateway to receive each booth microphone sensor reading for every 2s. The WSN gateway device and the sensor platforms are running on TinyOS 2.0 based applications.

The operation of the Smart Booth Navigation application is as follows:

- 1) The GuideSystem is assigned with BoothNavigation role and looking for devices which consumes the GuideService.
- 2) Visitor1 and Visitor2 enter to the WiFi coverage area of the GuideSystem.
- 3) Visitor1 and Visitor2 locations are recorded by the GuideSystem for every 5s.
- 4) The GuideSystem assigns BoothNavigation role to the Visitor1 and Visitor2 and both devices will be part of the BoothNavigation smart space.

The GuideSystem will only allow the devices which have necessary certificates.

- 5) The GuideSystem receives updates of each booth microphone reading every 2s.
- 6) When Visitor1 requests booth availability information, the GuideSystem will send available booth number and the Visitor1 latest location to the Visitor1.
- 7) By receiving available booth number and its current location, the Visitor1 will be able to navigate to the booth suggested by the GuideSystem.
- 8) The Guidesystem can also send any events related announcement and its location to the “BoothNavigation” smart space member devices using data centric communication, in this case to the Visitor1 and Visitor2.
- 9) If the visitors are interested in the announced events, they can request navigation information from the GuideSystem.

Figure 6.1 shows the smart booth navigation operation flowchart and the interaction between the devices involved.[46]

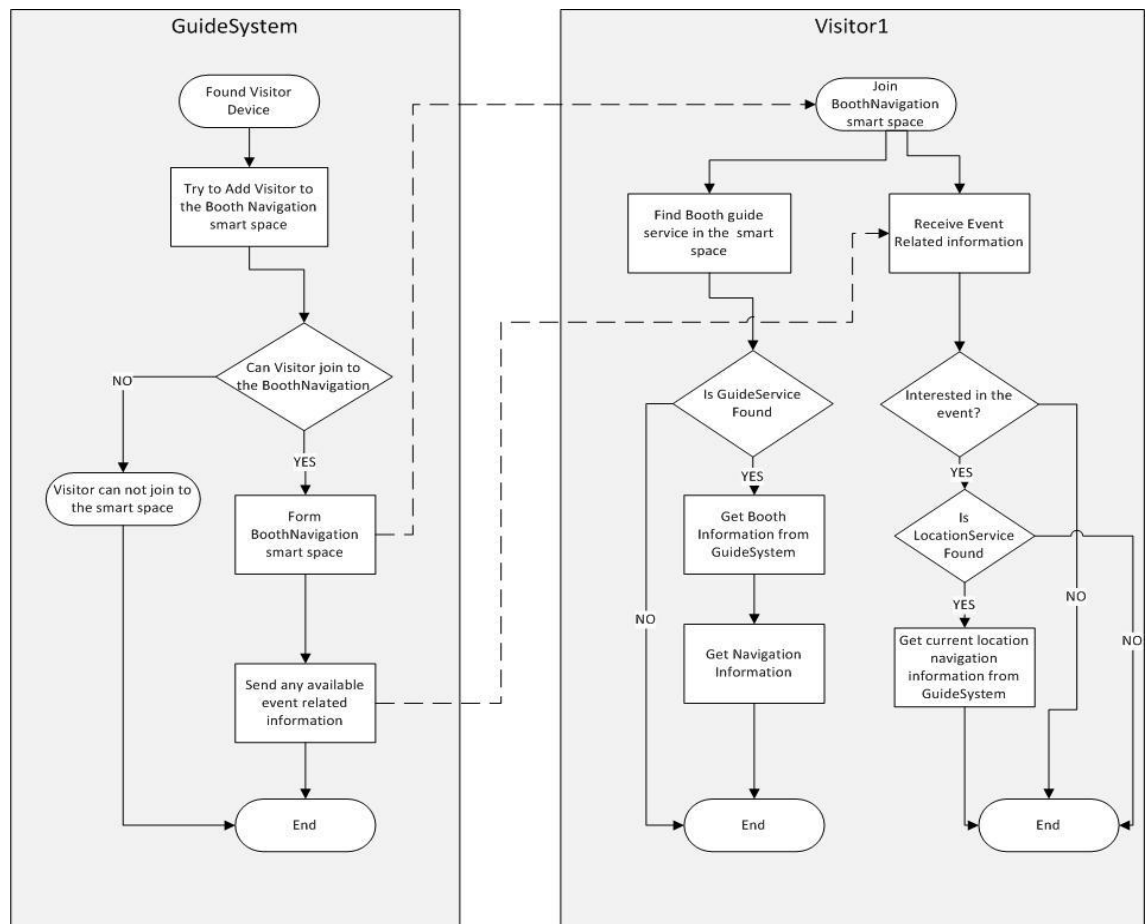


Figure 6.1: Smart booth navigation operation

6.2.1.2 Context

Table below shows the context of the three devices in this use case. The GuideSystem is the coordinator of the smart space and provides both the GuideService and the LocationService. Visitor1 and Visitor2 are both members of the smartspace and consume the services provided by the GuideSystem.

GuideSystem	
Type	Coordinator
Provides	GuideService
Provides	LocationService

Visitor1	
Type	Member
Carriedby	Visitor1
Consumes	GuideService
Consumes	LocationService

Visitor2	
Type	Member
Carriedby	Visitor2
Consumes	GuideService
Consumes	LocationService

Table 6.1: Smart Booth Navigation Context Details

6.2.1.3 Trust Chain

From the security perspective, the GuideSystem services are only available to registered users. For example, the Visitor1 and the Visitor2 are able to join the smart space with the GuideSystem, if three devices are installed with the certificates issued by a common authority (originated from a root certificate). Figure 6.2 shows a schema of the smart booth navigation trust chain.

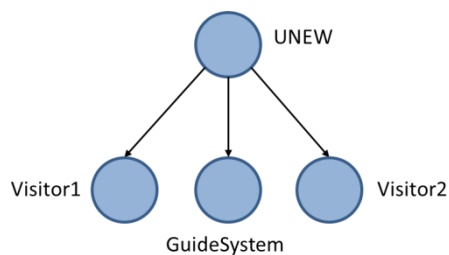


Figure 6.2: Smart booth navigation trust chain

Table 6.2 summarizes the trust levels and the certificates associated to each device involved in the smart booth navigation scenario.

Device	Trust level	Certificate
GuideSystem	Full	Certificate issued by Ran
	Marginal	-
	None	-
Visitor1	Full	Certificate issued by Ran
	Marginal	-
	None	-
Visitor2	Full	Certificate issued by Ran
	Marginal	
	None	

Table 6.2: Smart Booth Navigation Context Details

6.2.1.4 Role Specification

A BoothNavigation smart space is formed with the three devices described here based on the services they provide and services they consume. The GuideSystem acts as a coordinator and Visitor1 and Visitor2 take member role. The BoothNavigation role assignment is done by the following query by the GuideSystem.

```

PREFIX j.4: <http://www.ict-peces.eu/ont/smartspace.owl#>
PREFIX j.2: <http://www.daml.org/services/owl-s/1.1/Service.owl#>
SELECT ?device
WHERE
{
    ?device j.2:provides j.2:GuideService
}

PREFIX j.4: <http://www.ict-peces.eu/ont/smartspace.owl#>
PREFIX j.2: <http://www.daml.org/services/owl-s/1.1/Service.owl#>

```



```

SELECT ?device
WHERE
{
    ?device j.4:consumes j.2:GuideService
}

PREFIX j.4: <http://www.ict-peces.eu/ont/smartspace.owl#>
PREFIX j.2: <http://www.daml.org/services/owl-s/1.1/Service.owl#>
SELECT ?device
WHERE
{
    ?device j.2:provides j.2:LocationService
}

```

6.2.2 Development by Development Tools

Application developers will begin with the Configuration Tool and start configuring the application by drag and drop the devices in the workspace. Double clicking on the device will provide a new window where application developers can provide device properties. Different colour used to show the different type of devices.

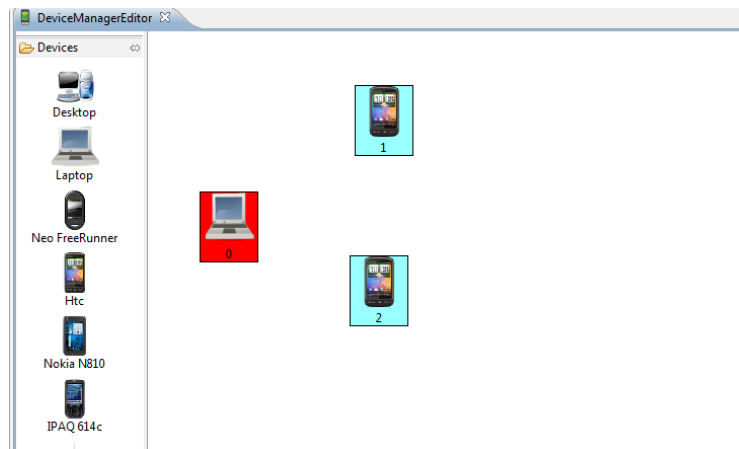


Figure 6.3: Booth Navigation: Device Definition Tool View

Initial context information, the application and services can be generated by Ontology Tool. As describe above GuideService and LocationService are provided by GuideSystem. Visitor1 and Visitor2 consume those services. The following screenshot shows the context definition by Ontology Tool.

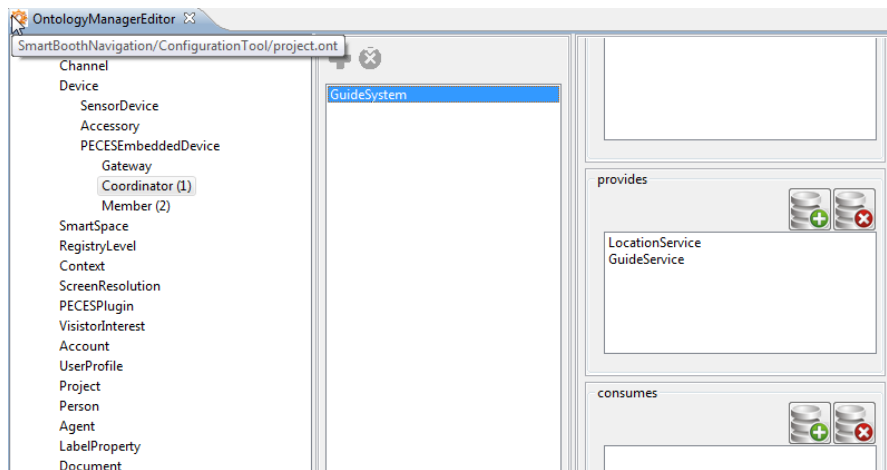


Figure 6.4: Booth Navigation: Ontology Definition Tool View

The necessary certificates will be deployed in a devices by the Configuration Tool for this scenario (assume I am the user of the Booth Navigation) and the figure 6.2 shows required certificate tree. Figure 6.5 shows the trust chain generated by Security Tool and files generated by security configuration tool.

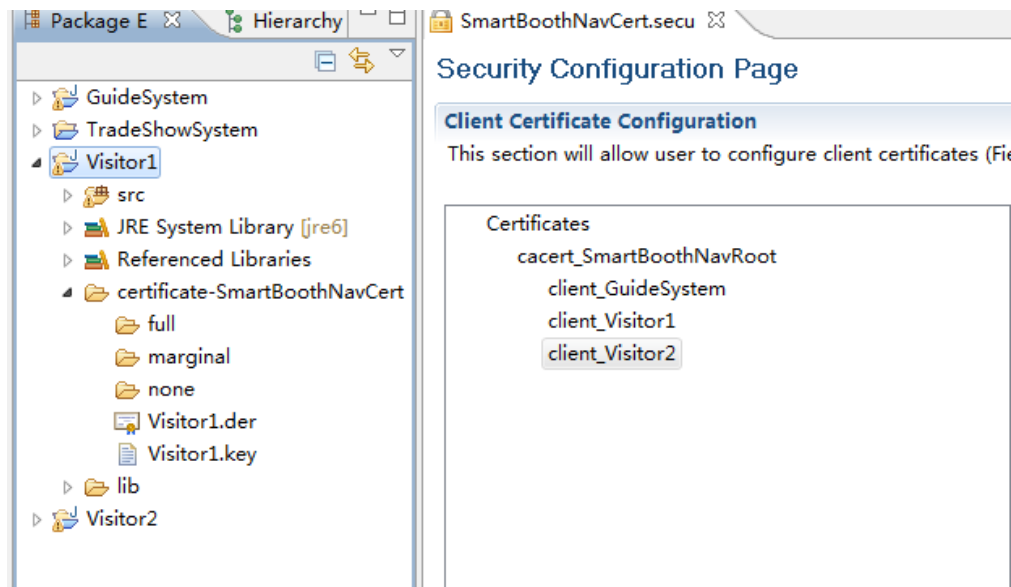


Figure 6.5: Booth Navigation: Security Tool View

The Configuration Tool output information then will be used by the Modelling Tool to model the actual test. Here the role specification will be defined which will form smart space with the devices configured by the Configuration Tool. If the application developers want to test the dynamics context, they will provide necessary information here. Developers will be able to provide different sensor reading values using the Modelling Tool. After the role speciation and complete

modelling process, smart space BoothNavigation formed with the configured devices as shown in the figures below.

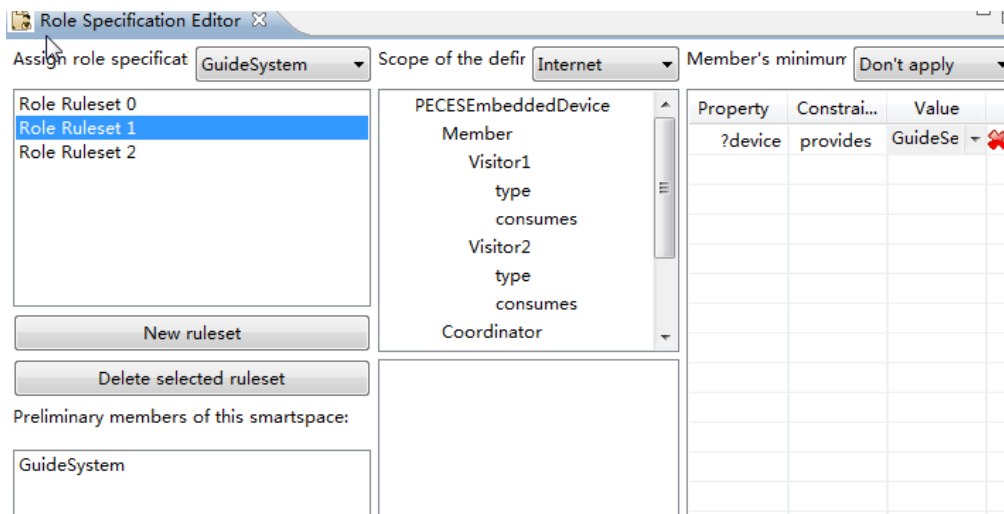


Figure 6.6: Smartspace Role Specification Modelling

6.2.3 Application Prototype

Three devices are used in this prototype scenario, namely the GuideSystem, the Visitor1 and the Visitor2.

- The GuideSystem is a Laptop (Windows XP) with WiFi and Internet connection. The GuideSystem is also connected with MIB520 programming board which attached with a Micaz mote[30]. It is also running Ekahau positioning engine software [54] to track the Visitor1 and Visitor2 location.



Figure 6.7: Guidesystem Laptop

- The Visitor1 is a HTC Sensation device running on Android 2.3.3 –see Figure 6.8.
- The Visitor2 is a HTC Desire device running on Android 2.2 –see figure 6.9.



Figure 6.8: HTC Sensation



Figure 6.9: HTC Desire

The WSN gateway connected with the GuideSystem receives each booth microphone sensor readings for every 2s. Figure 6.10 show the microphone sensor readings gathered from the wireless sensor platform installed in each booth at a particular time.

Microphone Sensor Reading		
Booth ID	Location	Mic
Booth8	(1347, 800)	461
Booth7	(950, 800)	464
Booth2	(685, 2015)	486
Booth3	(685, 2400)	464
Booth1	(180, 2200)	489
Booth4	(950, 1550)	899
Booth5	(1347, 1550)	463

Figure 6.10: Microphone sensor reading

The following screenshot shows the Visitor1 and the Visitor2 locations on a trade system local map when both devices entered in to the WiFi coverage area of the GuideSystem.

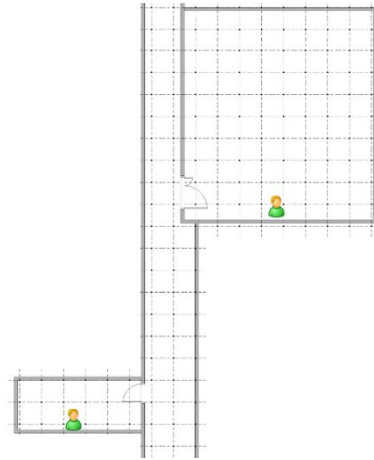


Figure 6.11: Visitor's location

When the Visitor1 and the Visitor2 entered to the GuideSystem coverage area, they joined with the BoothNavigation smart space as member devices with the GuideSystem as a coordinator. The screenshot shown in Figure 6.12 shows that the Visitor1 and Visitor2 joined with the BoothNavigation smart space and it also shows each device local location coordinates and SystemID.

Visitors joined with BoothNavigation Smart Space		
Visitors	Device ID	Location
Visitor2	7d82fdbd7da3b15f3f7a4f75ac1a5fc521aa5666	(196,2303)
Visitor1	cbf4eac1d9f133c0ddcac4e0dbedbc39ed3f4804	(1084,1373)

Figure 6.12: Visitors joined with the booth navigation smart space

Using the service centric communication mechanism, Visitor1 can request booth guide information from the GuideSystem. The screenshot in Figure 6.13 shows an interface provided to the Visitor1 to request available booth to visit.

Following the request from the Visitor1, the GuideSystem identified Booth8 as an available booth based on the all booth microphone readings and visitors location. The screenshot shown in Figure 6.14 shows the booth guidance information received by the Visitor1.

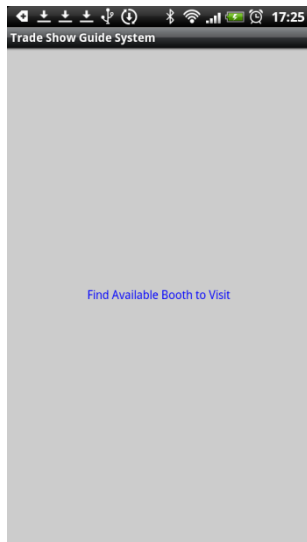


Figure 6.13: Available booth to visit

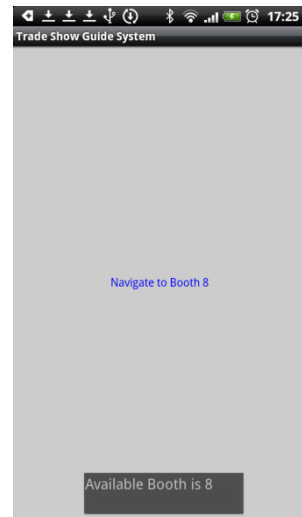


Figure 6.14: Booth guidance information

By receiving booth guidance information from the GuideSystem, the Visitor1 can navigate to Booth8. The screenshot shown in Figure 6.15 shows that Booth8 is shown to the Visitor1 on a local map.

The GuideSystem can also send any announcement to the BoothNavigation smart space members using data centric communication, for example sending information about an event in Room0 in 15 minutes. By receiving this information, visitors can navigate to the event location. The screenshot shown in Figure 6.16 shows a message received by the Visitor1.

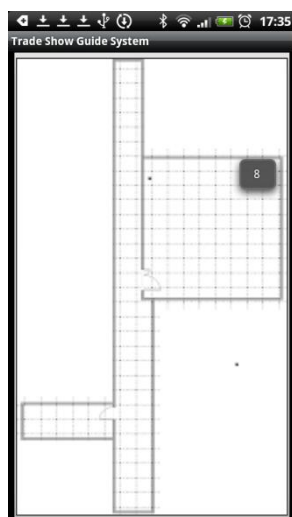


Figure 6.15: Available booth location

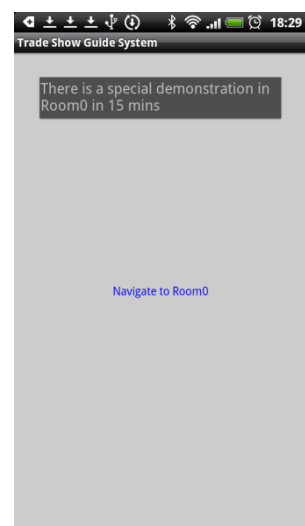


Figure 6.16: Additional information message

6.2.3 Application Limitations

This application has some limitations can be improved in future:

- All sensors' location should be defined before its deployed, system cannot collect it automatically
- Due to the wireless signal strength, the navigation system only works in around 10-15 metres. Extra wireless range booster may solve this problem
- Because using the old version of Ekahau in this application, sometimes the navigation is not accurate enough. A new professional version of Ekahau may improve this limitation.
- In this scenario, we hypothesize that booth with the lowest noise value is the available one for visitor. In real world, it may need more accurate way to define it.

6.3 Smart Booth Monitoring

The Smart Booth Monitoring is the second use case in the trade show guide system. This use case is an extend version of the Smart Booth Navigation use case. The objective is to provide features to handle any emergency situation at the trade show.

Smart Booth Monitoring use case consists of the Visitor1 and the Visitor2 and the GuideSystem as described in the previous section as well as another remote EmergencySystem. The EmengerncySystem is a Desktop PC (Windows 7) with internet connection.

When an emergency situation is detected at the trade show, the GuideSystem will be able to communicate with the remote EmergencySystem and provide necessary information about the incident for further action.

6.3.1 Scenario

6.3.1.1 Operation

The BoothNavigation smart space operation is the same as in the previous section, and the BoothNavigation smart space scope is global (available in the registry).

In this case, the GuideSystem takes both Coordinator and Gateway functionalities. Also, as mentioned before, the GuideSystem is providing another additional service: the EmergencyGuideService.

The EmergencySystem forms an Emergency smart space itself with global scope and takes both coordinator and gateway functionalities. When an emergency situation is detected at the trade show (based on the booth temperature and light sensor reading), the GuideSystem will form a hierarchical smart space with the BoothNavigation smart space (with the GuideSystem, Visitor1 and Visitor2) and the Emergency smart space (with EmergencySystem). After successfully assigned the hierarchical role assignment (called BoothNavigationEmergency), the GuideSystem will send emergency situation related messages to the all members of the hierarchical smart space, in this case (GuideSystem, Visitor1, Visitor2 and EmergencySystem).

The operation of the Smart Booth Monitoring application is as follows:

- 1) The BoothNavigation smart space operates as described in the previous section. The BoothNavigation smart space has global scope.
- 2) The EmergencySystem forms an Emergency smart space with global scope.
- 3) The GuideSystem receives updates of each booth temperature and light sensor reading for every 2s.
- 4) When an emergency situation is detected, the GuideSystem will inject a hierarchical role specification by including both BoothNavigation and Emergency smart spaces and will form a new hierarchical BoothNavigation Emergency smart space.
- 5) Using the data centric communication, the GuideSystem will send trade show GPS (pre-measure) location as well as booths location information

where emergency is detected to all members of the BoothNavigationEmergency hierarchical smart space.

- 6) The EmergencySystem will receive the emergency situation message with GPS location and local location information which will enable to the EmergencySystem to deal with the emergency situation efficiently.
- 7) Visitor1 and Visitor2 will receive the same message about the incident and also able to request emergency exit navigation information from the GuideSystem.

The figure below shows the smart booth monitoring operation flowchart and the interaction of the devices involved.

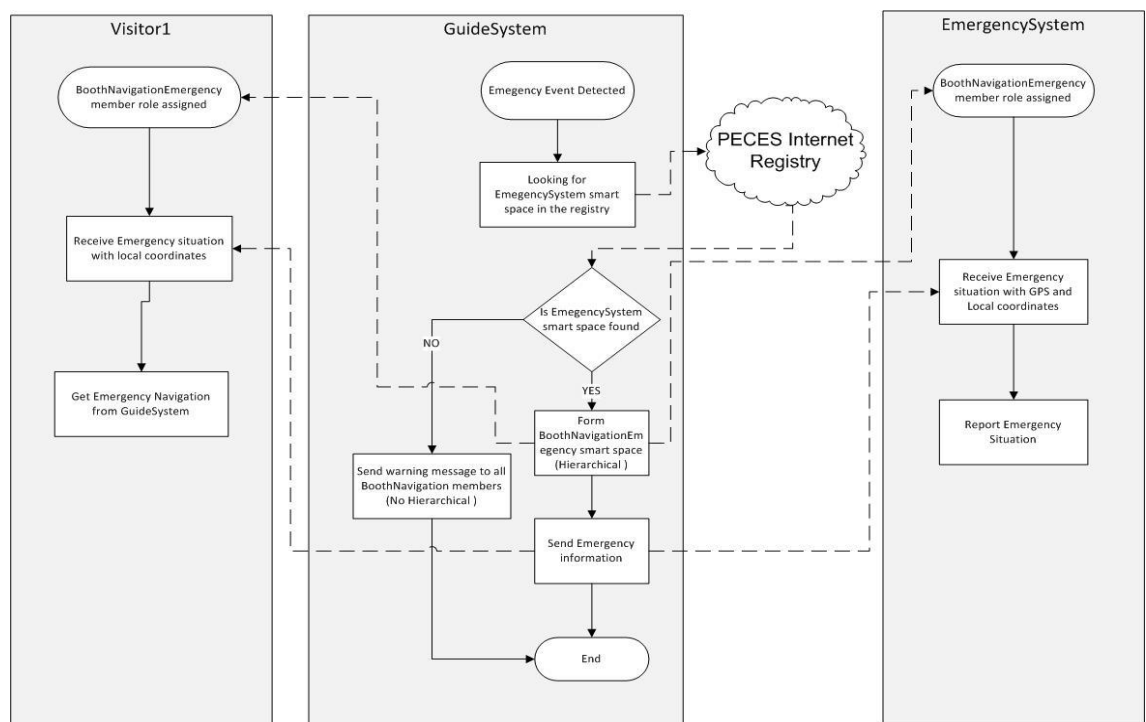


Figure 6.17: Smart booth monitoring operation

6.3.1.2 Context

Table below shows the additional context information of the BoothNavigation smart space member devices as well as the EmergencySystem. The GuideSystem is the coordinator of the BoothNavigation smart space, providing the EmergencyGuideService whilst the EmergencySystem is the coordinator of the Emergency smart space and provides the EmergencyService. Both Visitor1 and Visitor2 are member devices carried by the visitors and consuming the EmergencyGuideService.

GuideSystem	
Type	Coordinator
Provides	EmergencyGuideService

EmergencySystem	
Type	Coordinator
Provides	EmergencyService

Visitor1	
Type	Member
Carriedby	Visitor1
Consumes	EmergencyGuideService

Visitor2	
Type	Member
Carriedby	Visitor2
Consumes	EmergencyGuideService

Table 6.3: Smart Booth Monitoring Context Details

6.3.1.3 Trust Chain

Figure below shows a schema of the smart booth monitoring trust chain.

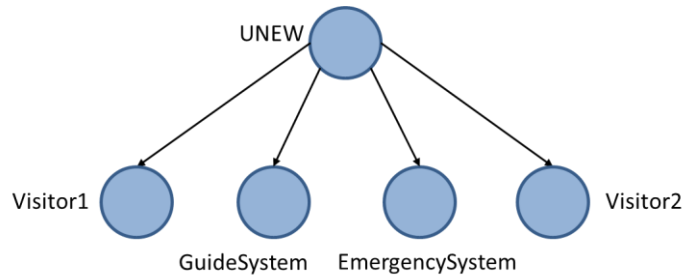


Figure 6.18: Smart booth monitoring trust chain

The trust relationship in this use case as shown in following table:

Device	Trust level	Certificate
GuideSystem	Full	Certificate issued by UNEW
	Marginal	-
	None	-
Visitor1	Full	Certificate issued by UNEW
	Marginal	-
	None	-
Visitor2	Full	Certificate issued by UNEW
	Marginal	
	None	
EmergencySystem	Full	Certificate issued by UNEW
	Marginal	
	None	

Table 6.4: Smart Booth Monitoring Trust Level and Certificates

6.3.1.4 Role Specification

The role specification of the BoothNavigation smart space is exactly the same as in the previous section. The EmergencySystem is the only one device that is part of the “Emergency” smart space and its role specification is carried by the following query:

```
PREFIX j.0: <http://www.daml.org/services/owl-s/1.1/Service.owl#>
SELECT ?device
WHERE
{
    ?device j.0:provides j.0:EmergencyService
}
```

6.3.2 Development by Development Tools

Application developers will start with the Configuration Tool by drag and drop the devices in the Configuration Tool workspace. By double clicking on the device, they will be able to provide the device configuration details. Here they will have to provide initial value for temperature to the Emergency System EmergencySystem (device ID 3).

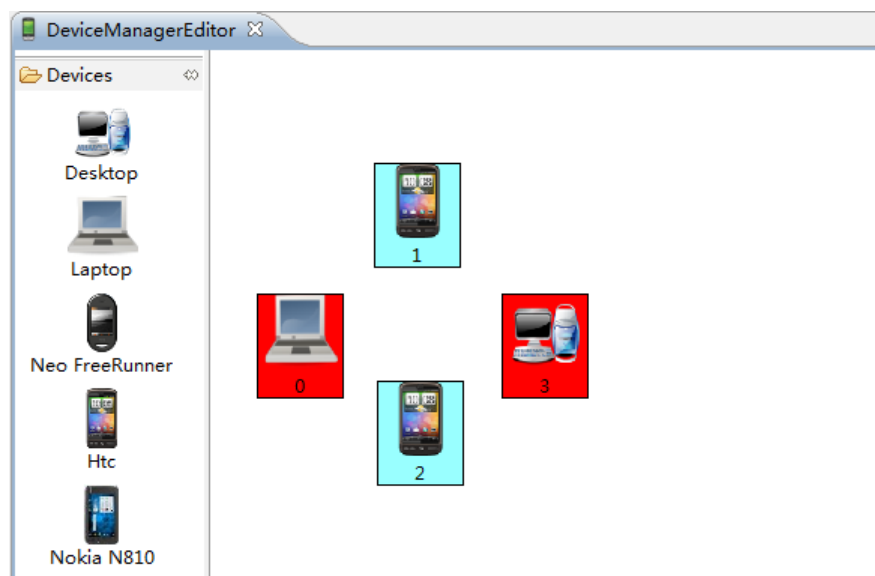


Figure 6.19: Emergency System: Device Definition Tool View

Initial context information, the application and services can be generated by Ontology Tool. As describe above EmergencyGuideService are provided by

both GuideSystem and EmergencySystem. Visitor1 and Visitor2 consume this service. The Figure 6.20 shows the context definition by Ontology Tool.

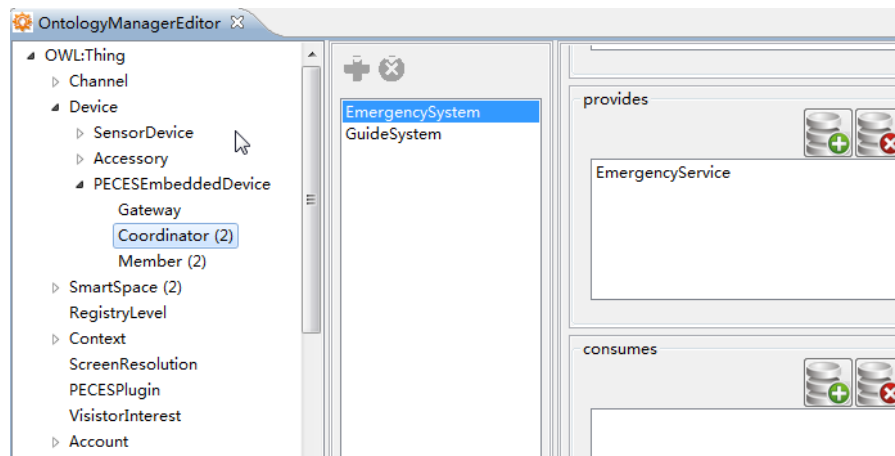


Figure 6.20: Emergency System: Ontology Definition Tool View

Necessary keys and certificates will be deployed to all devices by the openSSL tool which is part of the Configuration Tool. After providing all necessary configuration information to the devices, the Configuration Tool workspace will look like in Figure 6.21.

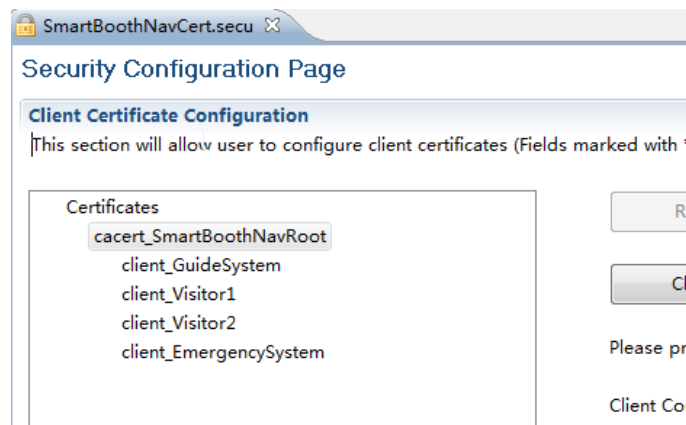


Figure 6.21: Emergency System: Security Tool View

The Modelling Tool will be used define the role speciation and dynamics in context, etc. Here dynamics context will be the temperature. Role specifications will defined based on the context (temperature) values. For example, if the context value (temperature) is below a certain value, Emergency smart space and BoothNavigation smart space will function as two separate smart spaces. If the context value is above certain value (high temperature value: emergency situation), Emergency smart space and BoothNavigation smart space will

merged and a new EmergencySys smart space will be formed (defined by the role specification). Figure 6.22 show the smart space EmergencySystem contained two devices GuideSystem and EmergencySystem.

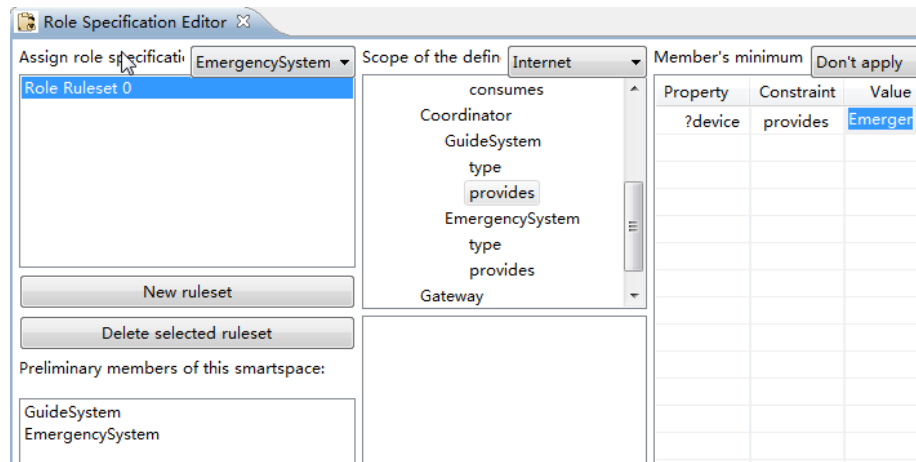


Figure 6.22: Emergency System: Role Specification Tool View

6.3.3 Application Prototype

This prototype is an extended version of the Smart Booth Navigation prototype described in the previous section. In addition to the three devices used in the Smart Booth Navigation prototype (GuideSystem, Visitor1 and Visitor2), another remote EmergencySystem device is involved in this prototype. The EmergencySystem is a Desktop PC (Windows 7) with internet connection.

The screenshot shown in Figure 6.23 shows the sensor readings gathered from the booth temperature and light sensors at a particular time.

Booth ID	Location	Temp	Light
Booth4	(950, 1550)	23	2294
Booth7	(950, 800)	26	2888
Booth8	(1347, 800)	27	3005
Booth5	(1347, 1550)	24	2318
Booth2	(685, 2015)	23	2208
Booth3	(685, 2400)	24	2135
Booth1	(180, 2200)	23	2224

Figure 6.23: Temperature and Light sensor readings

When an emergency situation is detected (higher value of light sensor reading at Booth7 and Booth8 in this scenario), the GuideSystem injects hierarchical role specification and sends its GPS location (pre-measured) as well as local location (in this case Booth7 location and Booth8 location). The message is received by the EmergencySystem as shown in Figure 6.24. Using this information the EmergencySystem can take effective action for this situation.

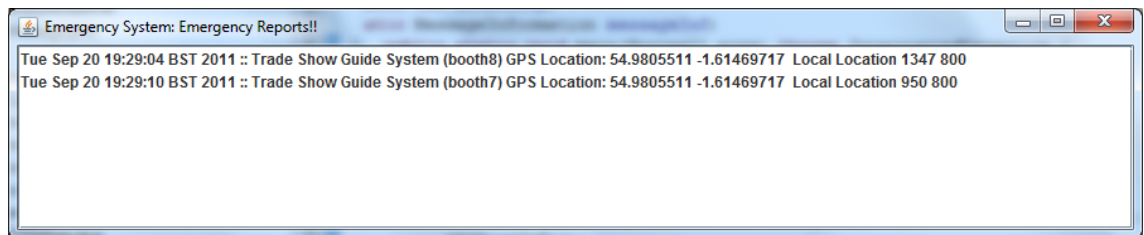


Figure 6.24: Emergency situation message received by the EmergencySystem

Finally, the Visitor1 and the Visitor2 also receive the same information but only local location information is displayed to the visitors. The Visitor1 and Visitor2 are able to get emergency navigation information from the GuideSystem. This shown in figure 6.25:

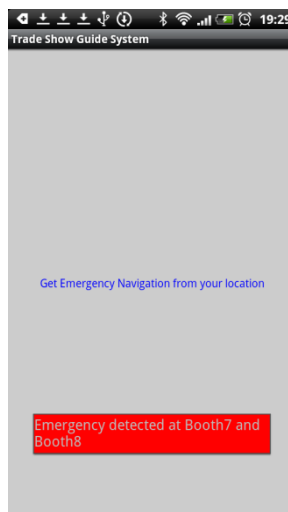


Figure 6.25: Information received by the visitors

6.3.3 Application Limitations

This application has some limitations can be improved in the future:

- In this scenario, we hypothesize that emergence event happened in the booth with the highest temperature value. In real world, it may need more accurate way to define it.
- The message sends by internet between emergency system and booth navigation system. It may cause problem when internet is shut down.

6.4 Smart Taxi Booking

The Smart Taxi Booking is the last use case in the trade show guide system. In addition to provide a seamless experience to the visitors at the trade show, the objective of this use case is to provide service to the visitors to book a taxi. This location aware booking service allows visitors to request a taxi without knowing his/her whereabouts in detail. Also the visitors do not have to wait for the taxi in the same location when they made request and can be able to move to another nearby location as the taxi service is able to track the visitor location changes.

6.4.1 Scenario

6.4.1.1 Operation

This prototype consists of two different devices: Visitor1 and Taxi1. Visitor1 is an Android based HTC Sensation Smartphone and Taxi1 is an Android based HTC Flyer Tablet. The Visitor1 is installed with PECES application with coordinator and gateway functionalities and it forms a TaxiBooking smart space by its own. The Taxi1 is installed with PECES application with coordinator and gateway functionalities and it forms a TaxiProviding smart space by its own. The operation of the Smart Taxi Booking application is as follows:

- 1) The Taxi1 forms a TaxiProvidingsmart space and it is made available to the internet registry.
- 2) The Visitor1 forms a TaxiBookingsmart space and it is made available to the internet registry.
- 3) Whenever a taxi service is needed, the Visitor1 forms a hierarchical smart space with TaxiBooking and TaxiProviding smart spaces and sends taxi request information with its GPS coordinate information.
- 4) When the request is received by the Taxi1, if it is willing to provide a service to the Visitor1, the Taxi1 forms a hierarchical smart space with

TaxiBooking and TaxiProviding smart spaces and sends response message with its latest GPS location information.

- 5) The Visitor1 is able to get update of the taxi location.
- 6) The Visitor1 moves to new location and this information is sent to the Taxi1.
- 7) The Taxi1 gets the Visitor1's new location and calculates the new route to get the Visitor1.
- 8) The Taxi1 reaches the Visitor1 at the new location.

The figure 6.26 shows smart taxi booking operation flowchart and the interaction of the devices involved.

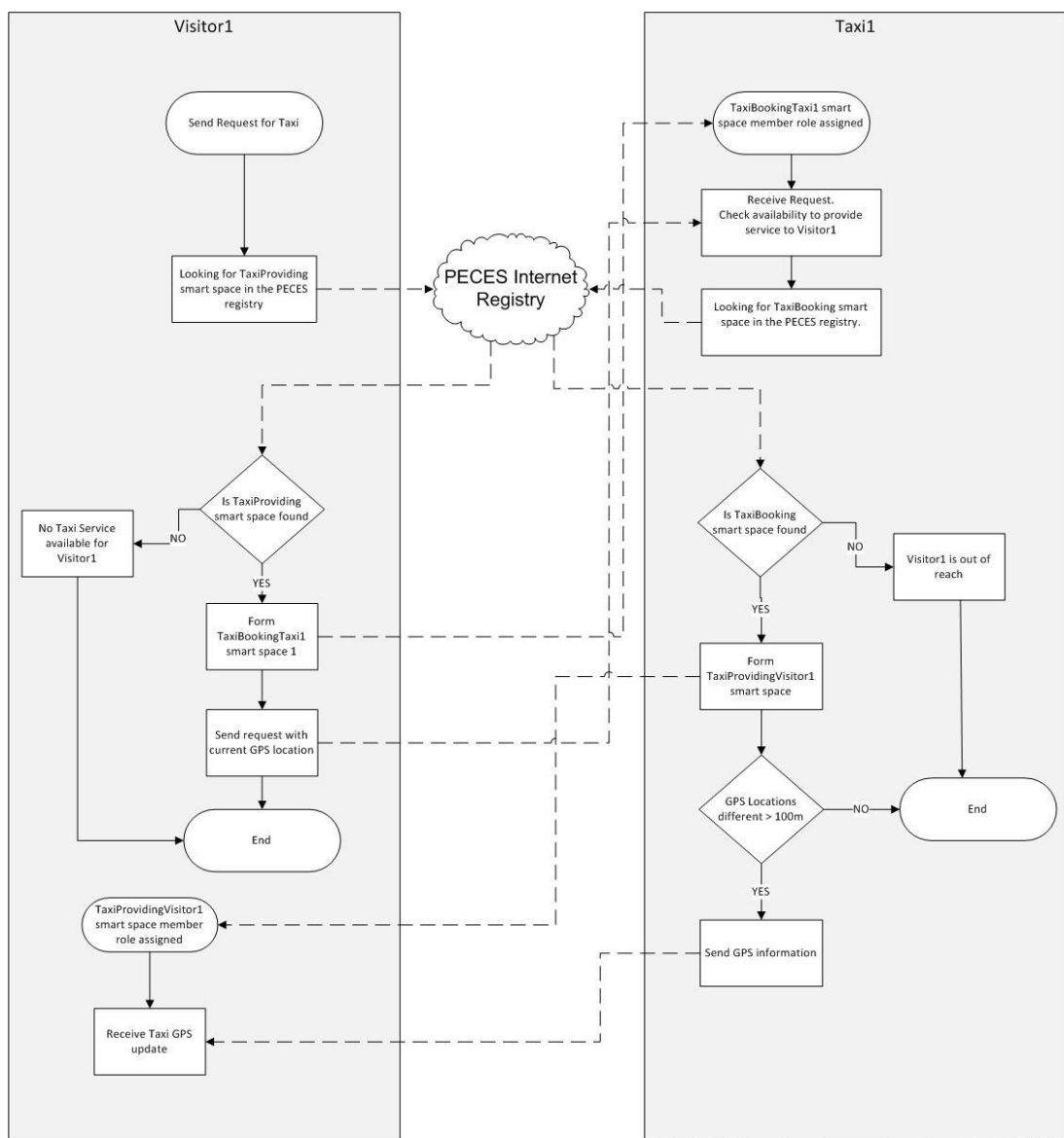


Figure 6.26: Smart taxi monitoring operation

6.4.1.2 Context

No context information is used in this prototype.

6.4.1.3 Trust Chain

The trust relationship in this use case as shown in Table 5

Device	Trust level	Certificate
Visitor1	Full	Certificate issued by UNEW
	Marginal	-
	None	-
Taxi1	Full	Certificate issued by UNEW
	Marginal	-
	None	-

Table 6.4: Smart Booth Monitoring Trust Level and Certificates

6.4.1.4 Role Specifications

The Visitor1 is assigned with TaxiBooking role and it alone forms a TaxiBooking smart space. The Taxi1 is assigned with TaxiProviding role and it alone forms a TaxiProviding smart space. When the Visitor1 wants to request a taxi, a hierarchical smart space (called TaxiBookingTaxi1) will be formed with TaxiBooking and TaxiProviding smart spaces. When the Taxi1 wants to send any message to the Visitor1, the Taxi1 will form a hierarchical smart space (called TaxiProvidingVisitor1) with TaxiProviding and TaxiBooking smart spaces.

6.4.2 Development by Development Tools

Application developers will start with the Configuration using drag and drop method to place the devices in the workspace. This application scenario will involve 2 devices. By double clicking on the device, developers will be able to provide device description information. Here they will have to provide sample GPS location to Visitor's PDA and Taxi driver's PDA. Another context value will be given here is the device mobility (Stationary, Non_Stationary) information.

Necessary keys and certificates will be deployed by the Configuration Tool too. After providing all necessary configuration information to the devices, the Configuration Tool will look like in the Figure 6.27 below.

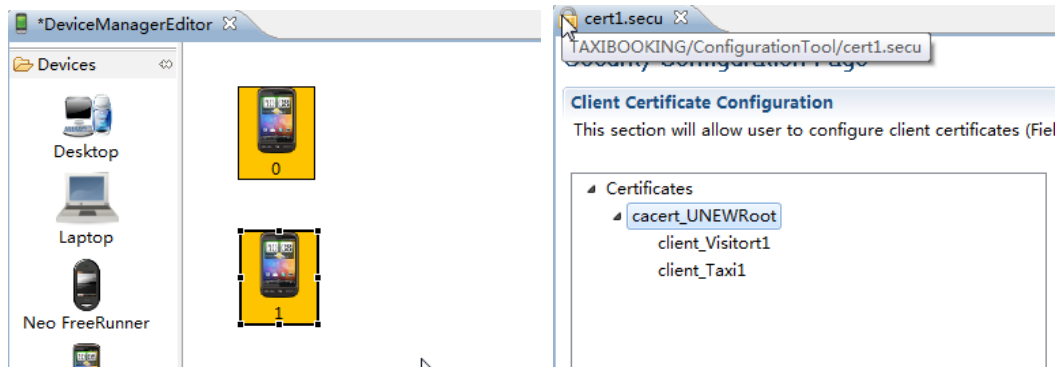


Figure 6.27: TaxiBooking System: Device Definition and Security Tool View

The next step is to use modeling Tool to identify role specification. The smart spaces TaxiProviding and TaxiBooking will be defined by role specification tool. As describe the above, a hierarchical smart space (called TaxiBookingTaxi1) will be formed with TaxiBooking and TaxiProviding smart spaces, which can be defined by Hierarchical Role Specification Tool. The figure below show the hierarchical smart space formed:

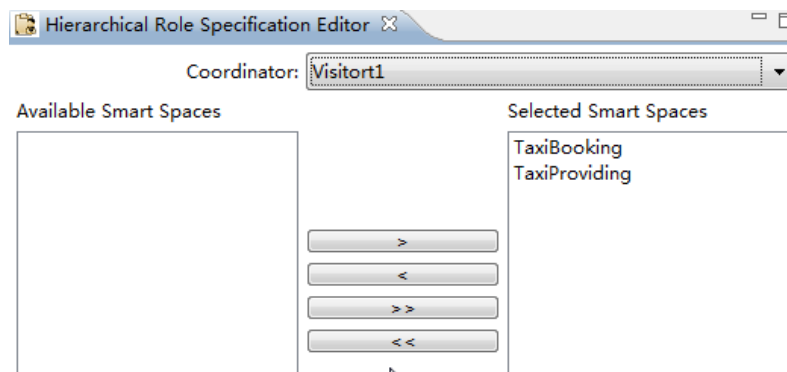


Figure 6.28: TaxiBooking System: Hierarchical Role Specification Tool View

6.4.3 Application Prototype

This prototype consists of two different devices: Visitor1 and Taxi1.

- The Visitor1 is a HTC Sensation Smartphone running on Android 2.3.3 –see Figure 6.17-.

- The Taxi1 is a HTC Flyer Tablet running on Android 2.3.3 –see Figure 6.18-



Figure 6.17: HTC Sensation Smartphone



Figure 6.18: HTC Flyer Tablet

When a TaxiBookingTaxi1 hierarchical smart space is formed with Visitor1 and Taxi1, Visitor1 sends its GPS location to the Taxi1 using data centric communication. Google Map API [53] is used to build default navigation map in this application.

When Taxi1 receives a request from the Visitor1, if the Taxi1 is available to provide a taxi service to the Visitor1, the Taxi1 will form TaxiProvidingVisitor1 smart space and sends a response with its GPS location to the Visitor1 using data centric communication. The screenshot shown in Figure 6.19 shows the initial location of both the Visitor1 and Taxi1 on the Visitor1's screen.

The screenshot shown in Figure 6.20 shows the initial route planning to Visitor1 location on the Taxi1's screen.

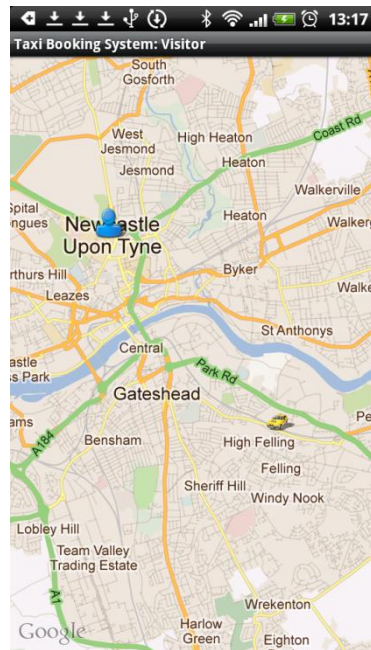


Figure 6.19: Initial location in the visitor1's device

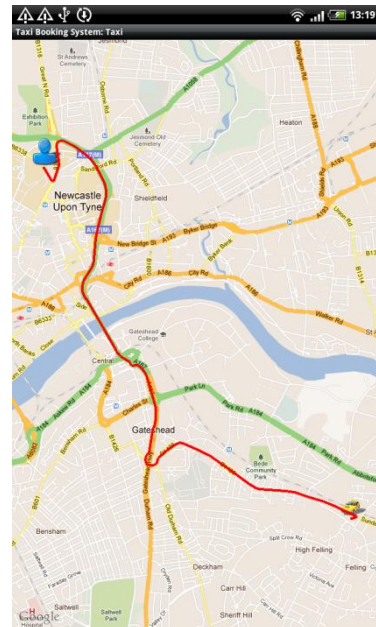


Figure 6.20: Initial route planning in the Taxi1 device

Since the Taxi1 keeps updating its GPS location, the Visitor1 is able to see current whereabouts of the Taxi1 shows in Figure 6.21. If any considerable changes occurred in the Visitor1 initial location (>100m between two GPS location updates), the Visitor1 sends its new GPS location to the Taxi1. The screenshot in Figure 6.22 shows that Visitor1 moved from its initial location to a new location.

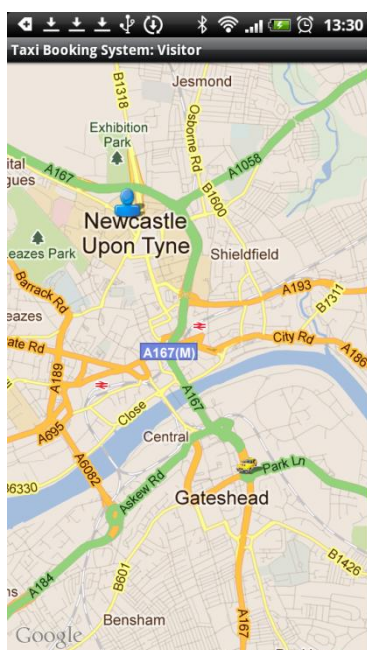


Figure 6.21: Information about the Taxi1 location

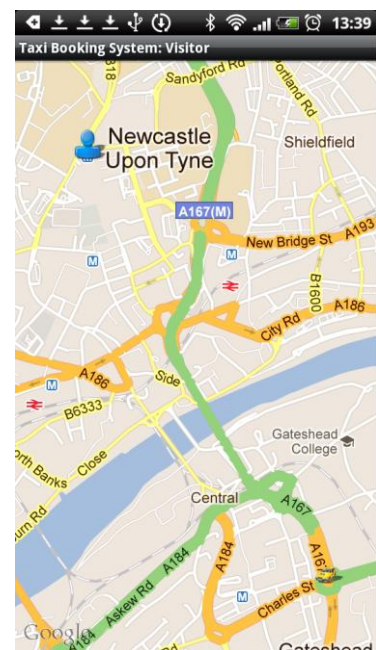


Figure 6.22: Visitor1's modification of location

When the new location information is received from the Visitor1, the Taxi1 gets a new route planning to the Visitor1 location. The screenshot in Figure 6.23 shows re-calculated route information on the Taxi1's screen.

Finally the Taxi1 reaches the Visitor1 new location and picks him/her up. The screenshot shown in Figure 6.24 shows the Taxi1 and the Visitor1 locations on the Visitor1's screen once the Taxi1 reached the Visitor1 new location.

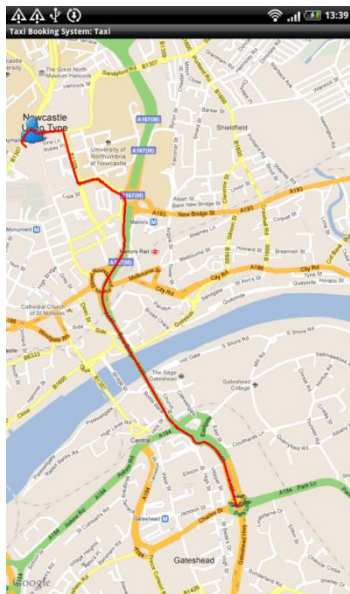


Figure 6.23: Taxi1 re-calculated route

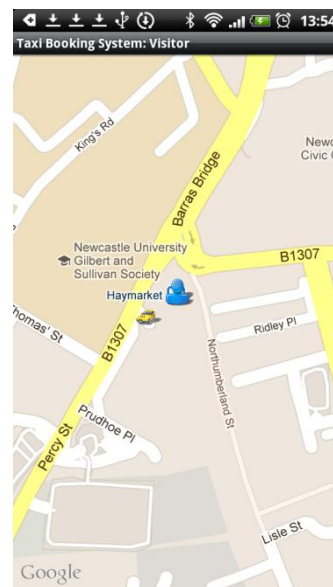


Figure 6.24: Taxi1 and Visitor1 final locations

6.4.3 Application Limitations

This application has some limitations can be improved in future:

- The current version of this application can only support visitor and taxi driver connection directly. The further improvement version may have a taxi centre to send the visitor's request to the nearest available taxi.

6.5 Summary

In this Chapter we presented a real application trade show system. There are three sub application are defined under trade show system. Each of them has its own features:

- Smart Booth Navigation: using single smart space Guidesystem to provide room information and navigation guide for visitors

- Smart Booth Monitoring: Using two smart spaces EmergencySystem and GuideSystem to form a Hierarchical smart space. It will send emergency information when emergency situation is detected.
- Smart Taxi Booking: Using two smart spaces Visitor and Taxi to form a Hierarchical smart space to provide a seamless way to booking a taxi.

All applications talked above can be supporting developed by Development Tools. Development tool dramatically decrease the difficulty and help the developers to build their own application without knowing the mechanism of the middleware.

Chapter 7 Evaluation

In order to verify whether Development Tools meet the hypothesis discussed in Chapter 1, there are some evaluations methods have been used to evaluate the Development Tools. A Laboratory testing evaluates the Development Tools requirements discussed in Chapter 1. A productivity plug-in is used to evaluate the Tools' performance. Because the end user of the development tools are the developers that want to create PECES applications, a questionnaire was handled to developers to get the real feedback from the people who will really use these tools.

Section 7.2 describe the evaluation methodological approach which will be use to evaluate the development tools. Section 7.3 lists the results of requirements evaluation. Section 7.4 shows the demonstration evaluation, includeing questionnaire design, result and analysis. This section also introduces the productivity plug-in analysis results.

7.1 Evaluation Summary

The PECES development tools simplify many of the steps for application development. For example, basic security related tasks such as the distribution of keys and certificates may be handled using the development tools.

The development tools suite provides features to build and test applications based on the PECES middleware. The tools are implemented as Eclipse plugins. The development tools suite provides several tools to support different activities during the application development, modelling and testing phases. There are configuration tools to assist the user with Device Definition, Ontology Instantiation, Security Configuration, Hierarchical Role Specification Definition and Service Definition. There is also an Event editor tool to dynamically model applications and a testing tool to test, analyse and then visual an application developed using the configuration tools.

7.2 Evaluation Methodological Approach

The objective of the PECES Development Tool evaluation process is to assess the performance of Development system functionalities in terms of complying with the technical and functional requirements. The relevant evaluation process

is used to evaluate the main objective of PECES development tools which is to facilitate the application development within the PECES middleware and will focus on the novel concepts developed by the PECES projects. The end user of the development tools are the developers that want to create PECES applications.

A methodological framework is proposed for the Development evaluation, taking into account the above features of the system. Under the proposed methodological framework the evaluation of the PECES system involves the following major steps: i) the determination of the objectives and the performance expectation for the Developments, ii) the identification of the measurable and not measurable requirements as well as any additional result issued, iii) identification of methods for assessing the achievement of requirements, iv) determination of the data needs and development of the relevant data collection tools, and v) data analysis and results. The figure 7.1 illustrates the overall methodological framework for the evaluation of the Development Tools results.

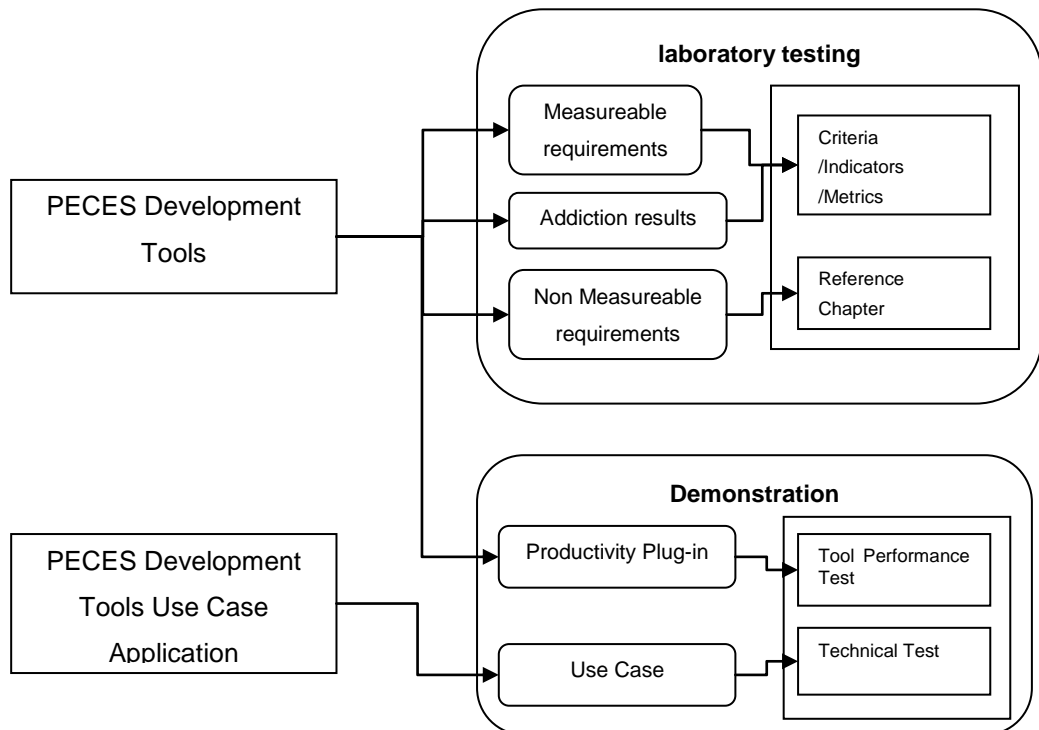


Figure 7.1: Development Tool Evaluation Framework

The evaluation expectations for the Development Tools emerge from the requirements specification performed within the design of the Development Tools in Chapter 1. The evaluation framework proposed is based on a laboratory testing phase where the Development Tools will be evaluated and a demonstration phase where the development tools and the use case applications will be assessed. The development tools evaluation is done both in the laboratory testing and in the demonstration phase following two different approaches.

The laboratory testing consists of the lab testing of the prototypes based on the requirements defined in Chapter 1. A previous classification of the requirements as measurable and not measurable has been done. For the measurable requirements, the relevant evaluation criteria, indicators and metrics are defined as well as the appropriate assessment methods and data needs. For the not measurable requirements, the reference documentation where the requirement achievement is justified is provided. The same approach is used for the additional results identified based on the fact that they are measurable or not.

The demonstration consists of the evaluation of the development tools and the assessment of the integration of the other three PECES prototypes into the applications. For the evaluation of the development tools, an ECLIPSE plug in is used as a monitoring tool which observes what the developer does and how long it takes him/her while he/she is working on a specific task. It is noteworthy to mention that the evaluation of the development tools is performed by developers as end users, as a discussion later. The evaluation of the integration of PECES prototypes in the applications is done based on a set of technical tests which already be discussed separately in Chapter 6.

The evaluation of PECES system is performed by developers, which are the end users of the PECES results. However, there are two different profiles that are involved in the evaluation of PECES: the developers that will perform the laboratory testing and the evaluation of the applications, and the developers that perform the evaluation of the development tools. The development tools will follow the basic procedure for the laboratory testing, but in the demonstration, they are tested by external developers that try to build a PECES project, creating a smart space with PECES development tools. These developers are

monitored by the productivity plug in which perspective or view they are using, what file has been saved, how many times, etc.-. Furthermore, they will complete a user acceptance questionnaire on the development tools to gather additional information for the system assessment.

7.3 List of Associated Requirements Evaluation Results

MEASUREMENT INSTRUMENT

Requirement	The development tools should support the programming language of the middleware
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The development tools are implemented as Eclipse plug-ins. Eclipse is a well known IDE for Java which is the the programming language of the middleware. The tools provided support to generate and build Java project for each devices which are participating in the smart space networks formation.

MEASUREMENT INSTRUMENT

Requirement	The development tools should provide support for the devices of the prototype applications
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The PECES middleware uses only the standard JAVA APIs which gives portability via Java Virtual Machine (JVM). Hence there is no necessity to deal with device specific elements. The development tools provided support for configuration of device communication capabilities and device functionalities. The development tools provided support for the devices used in the three prototpes applications where the tools enabled application developers to use drag and drop method to form networks and configure device functionalities and communication plug-ins.

MEASUREMENT INSTRUMENT

Requirement	The development tools should support the specification of policies to limit the distribution of context information
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The development tools provided mechanism for ontology instantiation and role specification which enable the application developers to limit the distribution of the context information.

MEASUREMENT INSTRUMENT

Requirement	The development tools should support the configuration of encryption keys
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	Current version of the development tools provided support to deploy keys and certificates as well as certificates trust chain for application development. The OpenSSL toolkit was integrated as an Eclipse plugin (Peces Security Configuration Tool) to generate/deploy keys and certificates for PECES middleware application development.

MEASUREMENT INSTRUMENT

Requirement	The development tools should support the specification of static device context
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The development tools provided support for specification of the context of the devices. Ontology instantiation tool has been integrated with Eclipse so that application developers do not

have to rely on external tools to this, for example, Protege.

MEASUREMENT INSTRUMENT

Requirement	The development tools should be integrated into an existing IDE
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The development tools are implemented as Eclipse plugins. Application developers are able to use not only the developments tools novel features but also able to use many features available in the Eclipse IDE.

MEASUREMENT INSTRUMENT

Requirement	The development tools should support the testing of group specifications
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The development tools provided support for the testing of the group specification based on the initial static context. Application developers can use the PECES role specification definition tool to define a smart space based on static context information defined in the ontology instantiation tool.

MEASUREMENT INSTRUMENT

Requirement	The development tools should support the modelling of a set of networked smart spaces
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The current version of the development tools prototype provided support for modelling local smart spaces based on switch on, switch off, context changes and connection changes of the

devices. The Event Editor provided a mechanism to generate different events with a delay time. The Event Diagram Editor allowed events to be organised and generates a xml file. The Testing Tool parsed the information in the xml file and generates necessary java project and code for emulating the defined model.

MEASUREMENT INSTRUMENT

Requirement	The development tool shall support debugging functionalities
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The Development tools provided support to generate Java project for each devices which are participating in the smart space formation. So that application developers can be able to use Java Eclipse debugging functionalities.

MEASUREMENT INSTRUMENT

Requirement	The development tool should support the graphical user interfaces of various devices and their interaction
Type	Not Measurable
Reference	Chapter 3, Chapter 4, Chapter 5
Result	Requirement Achieved
Discussion	The current version of the development tools prototype provided features to visualize the smart space devices, connections dynamics and context dynamics based on the test log data.

The Development Tools fulfil all the requirements identified by the Chapter 1 when their development. Furthermore the use of the tools has also been evaluated and was found to be extremely useful.

7.4 Developer Feedback

As mentioned in previous sections, the evaluation of the development tools will be made from two different perspectives: the productivity of the tools when developers use them and the opinion of the developers that will test them. For the first one, an ECLIPSE plug in will be used, as will be explained later on and for the second one, the following user acceptance questionnaire will be provided to the developers that will test the tools.

The PECES development tools have been evaluated by 20 evaluators from Germany, Spain and the UK. The evaluators were all Java programmers with varying degrees of experience ranging from Undergraduate Students thought to Post-Doctoral researchers and programmers from industry. Evaluators were given a short tutorial on PECES middleware and the development tools before testing the tools. They were asked to develop a simple service which required using the development tools (except for the Hierarchical Role Specification tool).

The evaluators completed a questionnaire and their development was measured by an Eclipse Productivity Plug-in.

There were two main purposes of this evaluation. The first was to establish that user found the PECES development tools useful application development. Secondly, the tool developers wanted to obtain some useful suggestions as to how to improve the tools in the future. *The Questionnaire shows in the Appendix 5.*

7.2.1 Questionnaire Result

The PECES development tools were evaluated by users in Germany in July 2011, in UK in October 2011 and in Spain in November 2011. Those in Germany were Undergraduate students, those in UK were Ph.D. students and those in Spain were company employees. This questionnaire was distributed to delegates to obtain their feedbacks. There were 20 delegates who participated in this evaluation from Germany, Spain and UK. The following table shows the result of the questionnaire.

Question	Result
<p>How difficult is to develop PECES middleware application without the PECES Development Tools ?(Range 1-5 where 1 indicated very difficult and 5 indicated very easy)</p>	<p>14 of 20 delegates answered this question, 5 delegates from Germany and 1 from Spain didn't answer this question.</p> <p>0 delegates select 5 0%</p> <p>0 delegates select 4 0%</p> <p>5 delegates select 3 36%</p> <p>5 delegates select 2 36%</p> <p>4 delegates select 1 29%</p> <p>Mean 2.07, Standard Deviation 0.82</p>
<p>What is the general impression you have for the PECES Development Tools? (Range 1-5 where 1 indicated very impressive and 5 indicated very unimpressive)</p>	<p>20 of 20 delegates answered this question.</p> <p>1 delegate selects 5 5%</p> <p>4 delegates select 4 20%</p> <p>3 delegates select 3 15%</p> <p>10 delegates select 2 50%</p> <p>2 delegates select 1 10%</p> <p>Mean 2.60, Standard Deviation 1.07</p>
<p>Indicate the level of training is required for the user to develop and test applications using the PECES Development Tools? (Range 1-5 where 1 indicated very low and 5 indicated very high)</p>	<p>20 of 20 delegates answered this question.</p> <p>1 delegate selects 5 5%</p> <p>3 delegates select 4 15%</p> <p>7 delegates select 3 35%</p> <p>8 delegates select 2 40%</p> <p>1 delegate selects 1 5%</p> <p>Mean 2.75, Standard Deviation 0.94</p>
<p>How reliable is the PECES Development Tools for the middleware application development? (Range 1-5 where 1 indicated very reliable and 5 indicated very unreliable)</p>	<p>20 of 20 delegates answered this question.</p> <p>1 delegate selects 5 5%</p> <p>3 delegates select 4 15%</p> <p>7 delegates select 3 35%</p> <p>8 delegates select 2 40%</p> <p>0 delegate select 1 0%</p> <p>Mean 2.9, Standard Deviation 0.889</p>
<p>How easy is to use the PECES Device Definition tool for the middleware application development? (Range 1-5</p>	<p>20 of 20 delegates answer this question.</p> <p>2 delegates select 5 10%</p> <p>1 delegate selects 4 5%</p>

where 1 indicated very easy and 5 indicated very difficult)	<p>3 delegates select 3 15%</p> <p>10 delegates select 2 50%</p> <p>4 delegates select 1 20%</p> <p>Mean 2.35, Standard Deviation 1.15</p>
How easy is to use the PECES Ontology Instantiation tool for the middleware application development? (Range 1-5 where 1 indicated very easy and 5 indicated very difficult)	<p>20 of 20 delegates answered this question.</p> <p>0 delegate select 5 0%</p> <p>2 delegates select 4 10%</p> <p>6 delegates select 3 30%</p> <p>11 delegates select 2 35%</p> <p>1 delegate selects 1 5%</p> <p>Mean 2.45, Standard Deviation 0.74</p>
How easy is to use the PECES Security Configuration tool for the middleware application development? (Range 1-5 where 1 indicated very easy and 5 indicated very difficult)	<p>12 of 20 delegates answered this question, 7 delegates from Germany and 1 from Spain didn't answer this question.</p> <p>0 delegate select 5 0%</p> <p>0 delegate select 4 0%</p> <p>7 delegates select 3 58%</p> <p>2 delegates select 2 17%</p> <p>3 delegates select 1 25%</p> <p>Mean 2.33, Standard Deviation 0.85</p>
How easy is to use the PECES Service Definition tool for the middleware application development (Range 1-5 where 1 indicated very easy and 5 indicated very difficult)	<p>20 of 20 delegates answered this question.</p> <p>0 delegate select 5 0%</p> <p>2 delegates select 4 10%</p> <p>4 delegates select 3 20%</p> <p>9 delegates select 2 45%</p> <p>5 delegates select 1 25%</p> <p>Mean 2.15, Standard Deviation 0.91</p>
How easy is to use the PECES Role Specification tool for the middleware application development? (Range 1-5 where 1 indicated very easy and 5 indicated very difficult)	<p>20 of 20 delegates answered this question.</p> <p>0 delegate select 5 0%</p> <p>6 delegates select 4 30%</p> <p>4 delegates select 3 20%</p> <p>6 delegates select 2 30%</p> <p>4 delegates select 1 20%</p> <p>Mean 2.60, Standard Deviation 1.11</p>

<p>How easy is to use the PECES Hierarchical Role Specification tool for the middleware application development? (Range 1-5 where 1 indicated very easy and 5 indicated very difficult)</p>	<p>14 of 20 delegates answered this question, 5 delegates from Germany and 1 from Spain didn't answer this question.</p> <p>0 delegate select 5 0%</p> <p>0 delegate select 4 0%</p> <p>9 delegates select 3 64%</p> <p>3 delegates select 2 21%</p> <p>2 delegates select 1 15%</p> <p>Mean 2.5, Standard Deviation 0.73</p>
<p>How easy is to use the PECES Event Definition tool and PECES Event Diagram tool for modelling smart space network dynamics? (Range 1-5 where 1 indicated very easy and 5 indicated very difficult)</p>	<p>15 of 20 delegates answered this question, 5 delegates from Germany didn't answer this question.</p> <p>0 delegate select 5 0%</p> <p>2 delegates select 4 13%</p> <p>3 delegates select 3 20%</p> <p>8 delegates select 2 54%</p> <p>2 delegates select 1 13%</p> <p>Mean 2.33, Standard Deviation 0.87</p>
<p>How easy is to use the PECES Testing Tool for testing smart space network application? (Range 1-5 where 1 indicated very easy and 5 indicated very difficult)</p>	<p>19 of 20 delegates answered this question, 1 delegate from Spain didn't answer this question.</p> <p>0 delegate select 5 0%</p> <p>0 delegate select 4 0%</p> <p>2 delegates select 3 11%</p> <p>11 delegates select 2 58%</p> <p>6 delegates select 1 31%</p> <p>Mean 1.79, Standard Deviation 0.61</p>
<p>Do you think that the PECES Development Tools are very useful for middleware application development and testing?</p>	<p>19 of 20 delegates answered this question, 1 delegate from Germany didn't answer this question.</p> <p>18 of 19 delegates answered yes. 95%</p> <p>1 of 19 delegates answered no. 5%</p> <p>94% delegates considered the PECES Development Tools are very useful for middleware application development and testing.</p>

How could the PECES Development Tools be improved?	<p>Improve Hotkeys and structure layout.</p> <p>Make the user interface more intuitive.</p> <p>Build in help manually in the tools.</p> <p>Provide clear messages when the user enters wrong or incomplete information.</p> <p>Some interface features can be improved.</p>
How do the PECES tools compare with other tools used to develop similar applications?	<p>Most delegates hadn't used this kind of tool before.</p> <p>Some of others thought these tools were very impressive and user friendly.</p>
Do you have any further comments relating to the usability of the PECES Development Tools	<p>It was suggested that the developers could get some ideas from Petri Net Tools.</p>

7.4.2 Questionnaire Result analysis

Figure 7.2 shows how difficult it was found to develop PECES middleware application without the PECES Development Tools. The range is 1-5 where 1 indicated very difficult and 5 indicated very easy. Developing applications on PECES middleware is thought to be difficult without development tools.

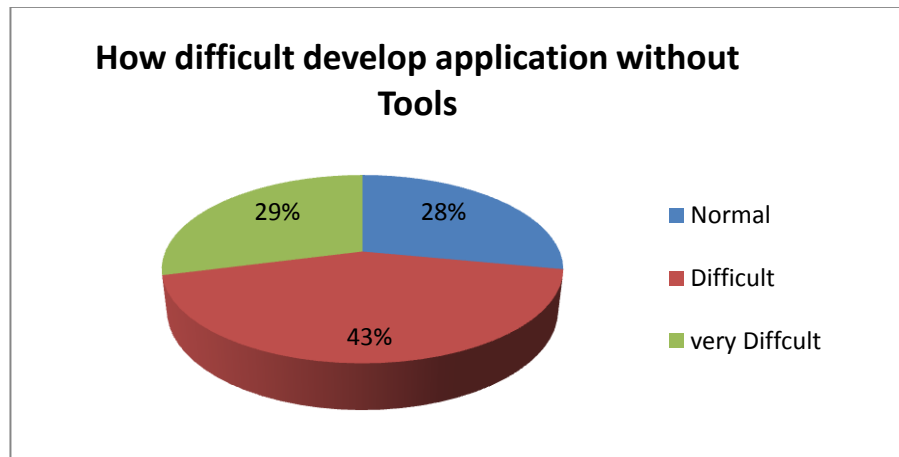


Figure 7.2: Difficulty of development without tools

The following figure indicates the responses to the question “what is the general impression you have for the PECES Development Tools”. Range 1 indicated very impressive and 5 indicated very unimpressive. Over half delegates (55%)

reported that the development tools are either impressive or very impressive. Only 25% people from the survey do not agree.

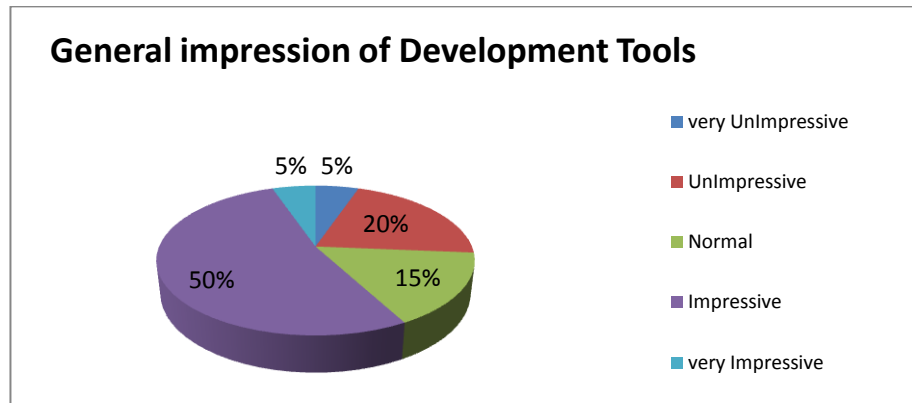


Figure 7.3: General impression of Development Tools

Figure 7.4 compares the ease of use of the different tools used. Every tool that was used is shown in the charts. The Range of responses was 1-5 where 1 indicated very easy to use and 5 indicated very difficult to use. Some tools are not considered easy to use because they require extra background knowledge or concepts from the middleware. The Ontology Definition tool, Role Specification tool and Hierarchical Role Specification tool were in this category.

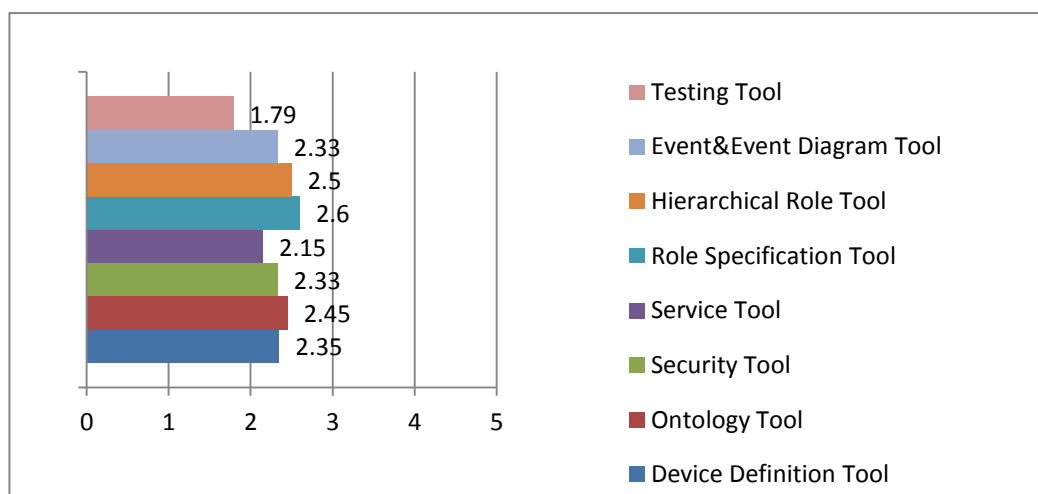


Figure 7.4: Mean of how easy to use development tools

Figure 7.5 depicts the percentage of users who agree that the PECES development tools are useful. 95% delegates believe development tools really

useful for developing PECES middleware based application. Only one person thought that the tool set is unsuitable for application development.

Finally, 55% of the evaluators believe a high level of training is needed whereas 40% think they just need a normal amount of training.

These results suggest that the PECES Development Tools are useful for experienced developers of PECES application rather than general users without any background knowledge of the middleware.

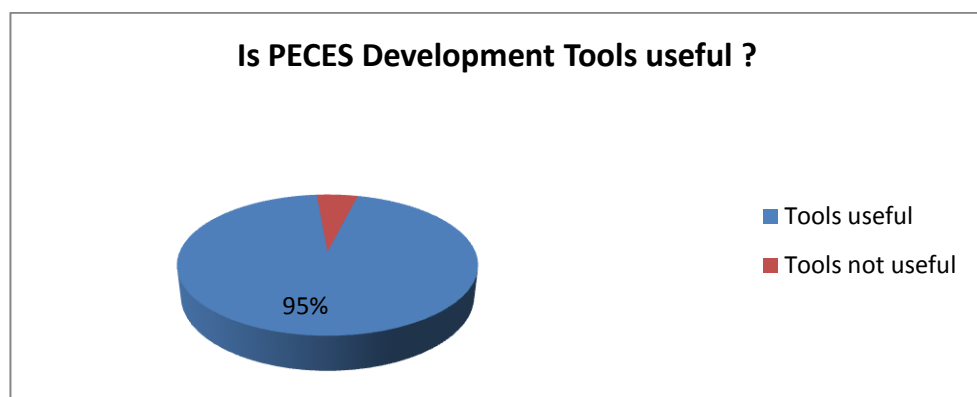


Figure 7.5: Are PECES Development Tools Useful

For HCI related, some of delegates thought these tools were very impressive and user friendly. Others give some suggestions for improve the tool in the future. Most the suggestions are list below:

- Improve hotkeys and structure layout: some delegates believe support hotkey can improve the usability for development tools
- Make the user interface more intuitive:
- Build in help manually in tools: We have a handbook for using development tools but some delegates think if there is a built-in help it may easier to looking for the resolution when meet some problems.
- Provide clear messages when the user enters wrong or incomplete information: someone think part of error messages are not enough clear to show how to find out the error
- Some interface features can be improved: such as role specification tool, it is a bit complex to use when delegates not enough familiar the tools.
- A delegate suggests that we can get some ideas form Petri Net.

7.4.3 Productivity Plug-in

For the development tools, statistical analysis was applied to the productivity log server.

The productivity log server was used to log the development related low level information (e.g. active file, perspective, developer, task, elapsed time, etc.). Each log was then uploaded to the productivity Log Server. The project manager estimates some project metrics such as the task time or the developers' experience and then, project metrics are calculated from the Productivity Log Server.

The metric used to evaluate these measurements was the time as an indicator of the effort required. The final objective was to obtain the assessment of the difficulty experienced by the developer or the learning curve with and without the use of the development tools until the developer is able to start producing applications with the middleware. The analysis of the data gathered with the productivity plug in together with the user acceptance questionnaires results will be used to extract these conclusions.

For performing the evaluation data gathering, a team of 5 developers was selected. The productivity plug in was installed on each workstation into the development environment to log the development related low level information. Each log was then uploaded to the productivity log server. This information was used to calculate the process metrics previously defined.

Another aim of the productivity plug-in connected to Eclipse is to continuously track the different tasks developers work on, the time they individually require for carrying out those tasks, and the file operations needed to be performed within a given development environment. During the monitoring process the plug-in saves which file the developer edited and saved, when they started an application server, and which project or sub-project the user was involved in, and in which perspective.

The plugin measures the user interaction and hence the difficulty of use of each tool. The Role Specification tool was found to be the hardest to use, requiring almost twice as many interactions as the other tools.

7.5 Summary

As discussed in this chapter, the Development Tools fulfil all the requirements identified by the Chapter 1 after laboratory testing evaluation. The questionnaires and productivity plug-in data prove the development tools are extremely useful for developers to build the smart space application base on PECES middleware. These results also show an experienced and highly educated developer finds the Development Tools much easier to use than general users without background knowledge of the middleware.

Chapter 8 Conclusions

Through all set of Development Tools has been shown and experimental use case are progressed their respective aims, It is worth revisiting their achievements, while also looking beyond to how the work could be progressed in the future. In this chapter, we describe the summary of the thesis in section 8.1. We also list the contributions and show the benefits to use the developments in section 8.2. The work maybe done in the future are list in the section 8.3.

8.1 Thesis Summary

The work described in this thesis is aimed to providing a set of tools to support and help the application developer to build applications using the PECES middleware and simulate the smart space dynamics such as device connections and context changes, etc. The development tools which are implemented as Eclipse plugins and integrated into the Eclipse Integrated Development Environment (IDE) can ease the middleware development process. The development tools provide graphical user interface (GUI) to configure, model and test the PECES middleware based smart space application.

There were a number of issues need to be considered and implemented during the development of the tools:

- Application developers require a set of tool to help them define devices which running in the application. They need to define the specification of the devices, such as name, communication plug-in, type, context, security trust chain, service provided, and role specification.
- If the application developer want to simulate and validate their applications before deploy it in real devices. A set of event and their sequence should be defined to simulate a real life circumstances for the applications and the services.
- After the definition of devices, environment and event, to validate the application, the application should be real run and some output can be generated to analysis the application.

To address these issue, there are three different toolset was developed. As described in Chapter 3, the configuration tool was designed and implemented to solve the first issue. This toolset consist a series of components:

- Device Definition Tool: provides a GUI to help developer defined devices. There are several device icons in the toolbar where developers can use to configure the device. By right click developer is able to modify device name, type and communication plug-in. Different type will be shown as different colour.
- Ontology Definition Tool: provides a interface to define static context information relevant to the device and this information will be used by the PECES middleware context components during the model execution.
- Security Tool: provides mechanism to generate public and private keys for each device by using OpenSSL tool. It will also provide mechanism to generate certificates. Application developers are able to generate asymmetric and symmetric cryptography keys and certificates for their devices.
- Service Definition Tool: provides a simple interface to the developers that allow the automatic generation of all the code needed to instantiate and make use of a PECES-based service.
- Role Specification Definition Tool: provides an interface where developers can define the different rules that the application will use to dynamically form groups of collaborative devices.
- Hierarchical Role Specification Tool: provides a interface to define smartspaces hierarchically.

In chapter 4, we described how to define dynamic context information, event and sequence of event by using modelling tool. This toolset consist a series of components:

- Event Editor: is used to edit single event definitions. Type, Contributing Devices, Description and Duration (Delay) can be defined in the wizard and later altered on the Overview Page. If the type is Device Context Change or Connection, further definition need to be done in the Context and Connection Page.

- Event Diagram Editor: provide an easy way to define the sequence of the events. This sequence is used to be running in testing tool.

A testing tool is developed to support the application developer to execute, simulate and validate the application defined by the Configuration Tool and the Modelling Tool. The Testing Tool allows application developer to start, stop and suspend the application and provide evaluation of the application results in a preferred format. Testing Tool provide the Execution Engine, the Connection Engine, the Instantiation Engine and the Event Engine to execute the application. A log file will be generated as an output to developer to evaluation. Testing tool has a Visualization Page which provides a engine to analysis the log file and show the result of application running as a more intuitive approach. Testing Tool is described in Chapter 5.

In Chapter 6, I built a real application trade show system which has three sub systems, Smart booth navigation, Smart booth monitoring and taxi booking. Scenario of the application and how to build the application by using Testing Tool has been demonstrated.

At last in Chapter 7, we evaluate the Development tool by Laboratory Testing and Demonstration Evaluation. We collect the data by questionnaire and productivity plug-in in Eclipse. All data in evaluation prove the research of development tool meet the requirement and hypothesis we describe in first chapter. The evaluation result also shows the PECES Development Tools are useful for experienced developers of PECES application rather than general users without any background knowledge of smart space and the middleware.

8.2 List of Contributions and Benefits of Using PECES Development Tools

Application developers might be able to develop application without using the development tools but they cannot properly test their application without using the development tools provided by PECES consortium. It can be argued that using the development tools hugely reduce the development efforts and hide complexity provided by the PECES Middleware. The following sections highlight the benefits of using each tool by comparing with the steps that have to be taken by the developers to develop PECES middleware based applications without using the tools.

a) PECES General Project

This tool is used to generate a PECES General Project Tool with three different folders namely Configuration Tool, Modelling Tool and Testing Tool to store different configuration, modelling and testing related files. This step is necessary to provide a consistent interface between the tools. This project also enables the user to locate necessary device java projects in the Eclipse workspace, to copy necessary files generated by other tools and execute the devices by the Testing Tool.

b) PECES Device Definition

The PECES Device Definition Tool provides features for initial configuration of the devices which will be used in the application. Using this tool, developers can generate necessary device Java middleware projects with a few mouse clicks and this tool provide device image based visualisation and this device image is very useful during the testing and analysing phase of the application development. If they don't use this tool, developers should create four different java projects and write java code for each device with appropriate middleware functionalities and communication plug-ins. In this case, application developers are not only writing java code for each device but also need to have very extensive knowledge of the PECES middleware.

c) PECES Ontology Instantiation Definition

The PECES Ontology Instantiation Tool provides features for Ontology Instantiation for the smart space application. Using this tool, developers can automatically load defined devices and its functionalities from the project.xml file generated by the Device Definition Tool. Application developers can add other context properties such as Services, Smart Space, etc. for their specific application. Once the developers completed context ontology definition of their application, this tool will generate project.owl file as well as device related ontologies for each device. Without using this tool, application developers would have to use Protege or another available ontology instantiation tool to define devices and functionalities and generate a common owl file. They would then need to manually generate the device context information for each device (the .pctx files). This would be a hugely time consuming task without this tool. .

d) PECES Security Configuration

The PECES Security Configuration Tool provides features to easily generate certificates and keys for the devices selected for the applications. The interface gathers necessary information for root certificate, intermediate certificate (for trust chain) and client certificate in one place. All of the necessary process of generating, copying and naming the certificate is done by this tool. If the developers do not use this tool, they would have to use the OpenSSL command line interface and provide necessary commands step by step. They have to repeat this for root certificate, trust chain and client certificate for each device. Once they completed this process, they would need to copy necessary root certificate, trust chain and client certificate and keys to the appropriate folders of the device Java project. This is a hugely time consuming and extremely error prone task.

e) PECES Service Definition

The PECES Service Definition tool reduces the learning curve of the PECES middleware developer and accelerates the development process of a PECES application. Developers do only need to focus on what kind of services they want to offer, and their actual implementation. The tool takes care of generating the code necessary to instantiate the service, making it public via the PECES middleware to other smart space partners and automatically generating all necessary proxy classes to allow clients of the service to interact with it via PECES in a simple way.

f) PECES Role Specification Definition

Role Specifications are a key element in the development of a PECES application, since the grouping of devices in smart spaces will be determined by these definitions. In addition, Role Specifications are closely related to context ontologies, another key element with a relatively large learning curve.

The Role Specification tool is necessary, since developers are able to get the following benefits:

- They are able to get the greatest potential offered by the context ontologies with minor effort, since the Role Specification tool assists the

developer by showing just the necessary information and hiding most of the complexity.

- They are able to observe the preliminary results of the Role Specifications they are writing, as they make changes to the Role Specification under development. This fact can ensure a high rate of correctness in the behavior of the smart space formation process during the development phase, even before any simulation is run.
- Since all the code resulting from the definition of the Role Specifications is automatically generated, correctness is ensured and the development process is greatly speeded up.

g) PECES Hierarchical Role Specification

The possibility to work with Hierarchical Role Specifications is one of the latest and most advanced features offered by the PECES Middleware. This kind of Role Specifications defines super-smart spaces as unions of basic smart spaces. It is not as well used as the basic version, since its use is only foreseen in applications with very specific needs. Since the code necessary to work with these Role Specifications is not straightforward to write, developers can take advantage of the PECES Hierarchical Role Specification tool to generate it, just by describing which groups they need to join.

h) Event Editor

The dynamics definition itself would be much harder without this editor because plain XML formatted information needs to be passed to the Testing Tool. The Event Editor covers all kind of dynamic specification in a dynamic, drag and drop and “click only” manner. This helps greatly in defining a complex testing scenario and could help significantly in application validation and testing of the core functionality of an application.

i) Event Diagram Editor

The Event Diagram Editor can be used immediately after the developer has defined the needed dynamics with the Event Editor tool. This is a very simple graphical, graph visualization tool where a circuit-free graph can be defined where the nodes are events. Every event has its delay and can be reused multiple times. With the help of this and the previous tool the developer can

assemble complex and sophisticated simulations with a few clicks and drags, instead of dealing with manually defined scripts or XML data sets.

j) PECES Testing Tool

The Testing Tool provides support to execute, analyse and then visualise the application test configured by the previous tools. In addition to this, the tool provides a visualisation of the log data of the test. This visualisation enables developers to clearly understand the PECES middleware functionality and evaluate its performance on the defined scenarios. Based on this, they can easily reconfigure their application and test it again until they get required results. Without this tool, application developers may not be able to test the modeled smart space applications, device connections and device context changes.

8.3 Future Work

There are still some room for improvements for the current version of the development tools.

- At present, application execution, testing and validation is running in the Eclipse platform, all application devices are assumed and generated as a Java project. That means the device need support java application if developer want to deploy the application to device directly. As can be seen in Device Definition Tool, it shows several different device icons. Nowadays, lot of devices especially in mobile devices are using other operation system such as Android, windows, IOS, Symbian, etc. The future work of this part is design a mechanism to generate appropriate code for application due to which kind of device is chosen.
- As describe in section 3.4, security tool can generate essential certificate and key for the applications. However, this tool has been optimized to support X.509 presently. It cannot support other security format well in this version. In the future, an advanced security tool will give more options to developer by use openssl features.
- PECES middleware supports building applications with multiple numbers of smart spaces. Even though all current the Peces Development Tools provide support for building application with multiple smart spaces except

the Peces Testing Tool's Visualization Page only supports two smart spaces visualisation at the moment. We are currently looking at the different options to provide support for visualizing multiple smart spaces.

8.4 Summary

One of the main objectives of the PECES middleware is to provide a cooperation layer that enables seamless interaction and coordination among devices in and across smart spaces in a secure manner. This thesis presented a set of tools which provide support for context aware and secure pervasive computing based application development. The tools provide support for device configuration, ontology instantiation, security configuration and role specification. The tools also enable dynamic modelling of the network connections and context changes. Finally, the tools provide support to test the smart space application performance and visualise the test results. The feedback of the Development Tools suggests that tools are useful for smart space application development. The evaluators' comments have been considerate and integrated in the tools already.

References

- [1]. PECES Project, <http://www.ict-peces.eu>, last accessed June 2012
- [2]. EMMA Project, <http://www.emmaproject.eu>, last accessed December 2010
- [3]. K. Selvarajah, C. Shooter, L. Liotti and A. Tully: *Heterogeneous Wireless Sensor Networks for Transportation Applications*, International Journal of Vehicular Technology (Special Issue on Vehicular Ad Hoc Networks), Feb 2011
- [4]. R. Grimm, T. Anderson, B. Bershad, D. Wetherall, "A system architecture for Pervasive Computing", proceedings of the 9th ACM SIGOPS European Workshop, pp. 177-182, Denmark, September 2000.
- [5]. H. Chen, F. Perich, T. Finin, and A. Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In 1st Annual Int'l Conf. on Mobile and Ubiquitous systems:Networking and Services, Aug. 2004..
- [6]. Edd Dumbill. Finding friends with xml and rdf. In IBM developerWorks, XML Watch. xmlhack.com, June 2002.
- [7]. PECES Consortium, *PECES Addressing Scheme Specification*, Deliverable D.3.1, PAS, <http://www.ict-peces.eu>, last accessed June 2012
- [8]. PECES Consortium, *PECES Communication Mechanisms and Registry Interface Specification*, Deliverable D.3.2, PAS, <http://www.ict-peces.eu>, last accessed June 2012
- [9]. C. Becker, G. Schiele, H. Gubbels, K. Rothermel: *BASE - A Micro-broker based Middleware For Pervasive Computing*, In Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications, pp. 443-451, Fort Worth, USA, March 2003
- [10]. Eclipse IDE, *Eclipse Website*, <http://www.eclipse.org/>, June 2012
- [11]. B. Lagesse, M. Kumar, J. M. Paluska and M. Wright, *DTT: A Distributed Trust Toolkit for Pervasive Systems*, <http://www.ioc.ornl.gov/publications/lagesseDTT.pdf>, last accessed June 2012
- [12]. M. Roman and R. H. Campbell, *A Middleware-based Application Framework for Active Space Applications*, Proceedings of the ACM/IFIP/USENIX International Conference on Middleware, 2003
- [13]. M. Roman and R. Campbell, *Gaia: Enabling Active Spaces, 9th ACM SIGOPS European Workshop*, pp.229-234, September 2000
- [14]. PECES Consortium, *PECES Secure Middleware Specification*, Deliverable D 4.1, PAS, <http://www.ict-peces.eu>, last accessed June 2012
- [15]. J. Barton and V. Vijayaraghavan, UBIWISE, *A Ubiquitous Wireless Infrastructure Simulation Environment*, <http://www.hpl.hp.com/techreports/2002/HPL-2002-303.html>, last accessed June 2012
- [16]. H. Nishikawa, S. Yamamoto, M. Tamai, K. Nishigaki, T. Kitani, N. Shibata, K. Yasumoto and M. Ito, UbiREAL: Realistic Smartspace Simulator for Systematic Testing, Proceedings of the 8th International Conference on Ubiquitous Computing (UbiComp2006), LNCS4206, pp. 459-476, Sep. 2006.
- [17]. PECES Ontologies, <http://www.ict-peces.eu/ont/>, last accessed June 2012
- [18]. Productivity plug-in version 3 <http://www.ict-peces.eu>, last accessed June 2012

- [19]. D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste, “*Project Aura: Towards Distraction-Free Pervasive Computing*”, IEEE Pervasive Computing, vol. 1, no. 2, pp. 22-31, April-June 2002.
- [20]. B. Johanson, A. Fox, and T. Winograd, “*The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms*”, IEEE Pervasive Computing, pp. 67-74, April-June, 2002
- [21]. Chun-Feng Liao, Ya-Wen Jong and Li-Chen Fu, *Toward Reliable Service Management in Message-Oriented Pervasive Systems*, IEEE Transactions on Services Computing, July-Sept. 2011, Volume: 4 Issue: 3, pp. 183 - 195.
- [22]. Protégé website: <http://protege.stanford.edu/>, last accessed June 2012
- [23]. R. Zhao, K. Selvarajah and N. A. Speirs, “*Development Tools for Pervasive Computing in Embedded Systems (PECES) Middle ware*”, Proceedings of the International Conference on Wireless Information Networks and Systems (WINSYS), 2011
- [24]. OWL-S: Semantic Markup for Web Services. <http://www.w3.org/Submission/OWL-S/>
- [25]. WGS84 Geo Positioning: an RDF vocabulary http://www.w3.org/2003/01/geo/wgs84_pos
- [26]. FOAF Vocabulary Specification. <http://xmlns.com/foaf/spec/>
- [27]. Time Ontology in OWL. <http://www.w3.org/TR/owl-time/>
- [28]. SPARQL Query Language for RDF. <http://www.w3.org/TR/rdf-sparql-query/>
- [29]. RDF Premier. <http://www.w3.org/TR/REC-rdf-syntax/>
- [30]. Crossbow. <http://www.xbow.com/>
- [31]. SPARQL Query Results XML Format. <http://www.w3.org/TR/rdf-sparql-XMLres/>
- [32]. Serializing SPARQL Query Results in JSON <http://www.mindswap.org/~kendall/sparql-results-json/>
- [33]. CSV. http://en.wikipedia.org/wiki/Comma-separated_values
- [34]. C. Becker, M. Handte, G. Schiele, K. Rothermel: *PCOM - A Component System for Adaptive Pervasive Computing Applications*. In Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications, Orlando, USA
- [35]. M. Haroon, M. Handte and P. J. Marrón: *Generic Role Assignment: A Uniform Middleware Abstraction for Configuration of Pervasive Systems*, In Proceedings of PerWare Workshop at the Seventh Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2009), March 2009
- [36]. CCITT, Recommendation X.509, “The Directory-Authentication Framework”, Geneva, 1989
- [37]. Anna V. Zhdanova and Uwe Keller: *Choosing an Ontology Language*
- [38]. M. Sintek, S. Decker, “TRIPLE - An RDF Query, Inference, and Transformation Language”, DDLP'2001, Japan, October 2001.
- [39]. M. Sintek, S. Decker, “TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web”. International Semantic Web Conference (ISWC), Sardinia, June 2002.
- [40]. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition): <http://www.w3.org/TR/owl2-syntax/>
- [41]. OWL Web Ontology Language Overview: <http://www.w3.org/TR/owl-features/>
- [42]. Apache Jena: <http://jena.apache.org/>
- [43]. RDF Schema RDFS: <http://www.obitko.com/tutorials/ontologies-semantic-web/rdf-schema-rdfs.html>
- [44]. Open Knowledge Base Connectivity Home Page: <http://www.ai.sri.com/~okbc/>

- [45]. Openssl home page: <http://www.openssl.org/>
- [46]. D. Pilone, N. Pitman: *UML 2.0 in a Nutshell*, 2nd Edition, O'Reilly Media, June, 2005
- [47]. A.D. Joseph, J.A. Tauber, and M.F. Kaashoek, "Mobile Computing with the Rover Toolkit", *IEEE Transactions on Computers: Special issue on Mobile Computing*, vol. 46, no.3, pp. 337-352, March 1997
- [48]. T. Kindberg, A. Fox, "System Software for Ubiquitous Computing", *IEEE pervasive computing*, vol. 1, no. 1, pp.70-81, January-March 2002
- [49]. G. Biegel, and V. Cahill : "A Framework for Developing Mobile, Context-aware Applications", *Proc. of 2nd IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom2004)*, pp. 361–365, 2004.
- [50]. T. Yamazaki, H. Ueda, A. Sawada, Y. Tajika, and M. Minoh : "Networked Appliances Collaboration on the Ubiquitous Home", *Proc. of 3rd Int'l Conf. on Smarthomes and health Telematic (ICOST 2005)*, Vol. 15, pp. 135–142, 2005.
- [51]. E. Chan, J. Bresler, J. Al-Muhtadi, and R. Campbell : "Gaia Microserver: An Extendable Mobile Middleware Platform", *Proc. of 3rd IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom2005)*, pp. 309–313, 2005.
- [52]. K. Sanmugalingam, and G. Coulouris : "A Generic Location Event Simulator", *Proc. of 4th Int'l Conf. on Ubiquitous Computing (UbiComp2002)*, pp. 308–315 ,2002.
- [53]. Google Map API:
<https://developers.google.com/maps/documentation/staticmaps/index?hl=zh-en>
- [54]. Ekahau: <http://www.ekahau.com/>
- [55]. Jerry R. Hobbs. A daml ontology of time.
<http://www.cs.rochester.edu/~ferguson/daml/daml-time-20020830.txt>, 2002.
- [56]. Feng Pan and Jerry R. Hobbs. Time in owl-s. In Proceedings of AAAI-04 Spring Symposium on Semantic Web Services, Stanford University, California, 2004.
- [57]. Douglas B. Lenat and R. V. Guha. Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project. Addison-Wesley, February 1990.
- [58]. David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning, 1992.
- [59]. Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 2003.
- [60]. Filip Perich. MoGATU BDI Ontology, 2004.
- [61]. Lalana Kagal, Massimo Paolucci, Naveen Srinivasan, Grit Denker, Tim Finin, and Katia Sycara. Authorization and privacy for semantic web services. AAAI 2004 Spring Symposium on Semantic Web Services, March 2004.
- [62]. X.H. Wang, D.Q. Zhang, T. Gu, and H.K. Pung. Ontology-Based Context Modeling and Reasoning using OWL. In Context Modeling and Reasoning Workshop at PerCom, pp. 18–22, March 2004.
- [63]. D. Preuveneers, J. v.d.Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. In 2nd European Symposium on Ambient Intelligence, Nov. 2004.
- [64]. N. Belhanafi, Ch. Taconet, and G. Bernard. CAMidO, A Context-Aware Middleware Based on Ontology Meta-Model. In Workshop on Context Awareness for Proactive Systems, pp. 93–103, June 2005.
- [65]. Resource Description Language. <http://www.w3.org/RDF/>.
- [66]. RDF Vocabulary Description Language 1.0: RDF Schema.
<http://www.w3.org/TR/rdf-schema/>.

- [67]. SWRL: A Semantic Web Rule Language Combining OWL and RuleML.
<http://www.w3.org/Submission/SWRL/>.
- [68]. Reto Krummenacher, Holger Lausen, Thomas Strang. Analyzing the Modeling of Context with Ontologies. Int'l Workshop on Context-Awareness for Self-Managing Systems
- [69]. Michael H. Coen. Design principles for intelligent environments. In Proceedings of AAAI/IAAI 1998, pages 547–554, 1998.
- [70]. J.W. Lloyd. Foundations of Logic Programming. Springer, 1987.
- [71]. J. Kohl, C. Neuman “The Kerberos Network Authentication Service (v5)”, IETF RFC 1510, September 1993.
- [72]. Bresson, O. Chevassut, D. Pointcheval, J. Quisquater, "Provably Authenticated Group Diffie-Hellman Key Exchange" Proceedings of the Eight ACM conference on Computer and Communications Security, 2001.
- [73]. R. Mayrhofer, H. Gellersen, M. Hazas, "Security by Spatial Reference: Using Relative Positioning to Authenticate Devices for Spontaneous Interaction", International Conference on Ubiquitous Computing, 2007.
- [74]. A. Varshavsky, A. Scannell, A. LaMarca, E. de Lara, "Amigo: Proximity-Based Authentication of Mobile Devices", International Conference on Ubiquitous Computing, 2007.
- [75]. Y. Zhang, W. Liu, W. Lou, and Y. Fang, "Securing Sensor Networks with Location-Based Keys," presented at IEEE Wireless Communications and Networking Conference (WCNC 2005), New Orleans, LA, 2005.
- [76]. D. Carman, P. Kruus, B. Matt, "Constraints and Approaches for Distributed Sensor Network Security", NAI Labs, Tech. Rep. #00-010, September 2000.
- [77]. A. Perrig, R. Canetti, J. Tygar, D. Song, “The TESLA broadcast authentication protocol”, RSA CryptoBytes, 2002.
- [78]. S. Kent, R. Atkinson, "Security Architecture for the Internet Protocol", IETF RFC 2401, November 1998.
- [79]. T. Dierks, E. Rescorla, "The Transport Layer Security (TLS) Protocol", IETF RFC 4346, April 2006.
- [80]. A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, J. D. Tygar, "SPINS: Security Protocols for Sensor Networks", Wireless Networks, vol. 8, pp. 189–199, 2002.
- [81]. C. Karlof, N. Sastry, D. Wagner "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks", Second International Conference on Embedded Networked Sensor Systems, pp. 162–175, New York, USA, 2004.
- [82]. S. Slijepcevic, M. Potkonjak, V. Tsiatsis, S. Zimbeck, M. Srivastava, "On Communication Security in Wireless Ad-Hoc Sensor Networks", Proceedings of the Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002.
- [83]. S. Basagni, K. Herrin, D. Bruschi, E. Rosti, “Secure PebbleNets”, Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2001.
- [84]. D. Liu, P. Ning, R. Li, "Establishing Pair-wise Keys in Distributed Sensor Networks", in ACM Transactions on Information and System Security, Vol. 8, No.1, pages 41-77, February 2005.
- [85]. S. Zhu, S. Setia, Sushil Jajodia, " LEAP: Efficient Security Mechanisms for Large-scale Distributed Sensor Networks", Centre for Secure Information Systems, George Mason University, ACM Press, pp. 62-72, 2003.
- [86]. R. Roman, J. Zhou, J. Lopez, "Applying Intrusion Detection Systems to Wireless Sensor Networks", Consumer Communications and Networking Conference (CCNC'06), pp. 640-644 IEEE Press, Las Vegas, USA. January 2006.

- [87]. J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, D. Ganesan, "Building Efficient Wireless Sensor Networks with Low-Level Naming", ACM Symposium on Operating Systems Principles, pp.146-159, 2001.
- [88]. ReoLive Project by "Centrum Wiskunde & Informatica - CWI"
<http://reo.project.cwi.nl/cgi-bin/trac.cgi/reo/wiki/NetworkIdentifiers>, last accessed February 2009.
- [89]. H. Baldus, K. Klabunde, G. Müsch, "Reliable Set-Up of Medical Body-Sensor Networks", Lecture Note in Computer Science, pp. 353-363, 2004.
- [90]. H. Fientein, R. Sandhu, E. Coyne, C. Youman, "Role Based Access Control Models", IEEE Computer, pp. 38-47, 1996.
- [91]. Sesame: RDF Schema Querying and Storage. <http://www.openrdf.org/>.
- [92]. L. Giuri, P. Iglío, "Role Templates for Content-Based Access Control" Proceedings of the Second ACM Workshop on Role Based Access Control, pp. 153-159, 1997.
- [93]. R. Thomas, "Team-Based Access Control (TMAC): A Primitive for Applying Role-Based Access Controls in Collaborative Environments", ACM Workshop on Role Based Access Control, 1997.
- [94]. A. Abou-El-Kalam, et al., "Organization Based Access Control", IEEE Fourth International Workshop on Policies for Distributed Systems and Networks, 2003.
- [95]. G. Zhang, M. Parashar, "Dynamic Context-aware Access Control for Grid Applications", International Conference on Grid Computing, 2003.
- [96]. A. Herzberg, Y. Mass, J. Mihaeli, "Access Control Meets Public Key Infrastructure or: As Singing Roles to Strangers", Proceedings of the IEEE Symposium on Security and Privacy, pp. 2-14, 2000.
- [97]. R. Hayton, J. Bacon, K. Moody, "Access Control in an Open Distributed Environment", Proceedings of the IEEE Symposium on Security and Privacy, pp. 3-14, 1998.
- [98]. S. Jajodia, P. Samarati, V. Subrahmanian, E. Bertino, "A unified Framework for Enforcing Multiple Access Control Policies", Proceedings of the 1997 ACM International SIGMOD Conference on Management of Data, May 1997.
- [99]. T. Woo, S. Lam, "Designing a Distributed Authorization Service", Processing of IEEE INFOCOM, 1998.
- [100]. Arun Mukhija and Martin Glinz, "The CASA Approach to Autonomic Applications", Proceedings of the 5th IEEE Workshop on Applications and Services in Wireless Networks (ASWN 2005), Paris, France, June-July 2005.
- [101]. Component Synthesis with Model Integrated Computing (CoSMIC), <http://www.dre.vanderbilt.edu/cosmic/>, last accessed February 2009.
- [102]. N. Wang, K. Balasubramanian, C. Gill, "Towards a real-time corba component model," in OMG Workshop On Embedded & Real-Time Distributed Object Systems, Washington, D.C., July 2002, Object Management Group.
- [103]. R. Vanegas, J. Zinky, J. P. Loyall, D. Karr, R. E. Schantz, and D. E. Bakken, "QuO's Runtime Support for Quality of Service in Distributed Objects," Proceedings of Middleware 98, the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing, September 1998.
- [104]. IST-MUSIC, <http://www.ist-music.eu/>, last accessed Feb 2011.
- [105]. The MADAM project, <http://www.ist-music.eu/MUSIC/madam-project>, last accessed Feb 2011.
- [106]. PLASTIC, <http://www.ist-plastic.org/>, last accessed Feb 2011.

Appendix 1 - project.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:projectSettings xmlns:tns="http://www.example.org/project"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <tns:general>
    <tns:projectName>/DEMOPROJECT</tns:projectName>
    <tns:projectFolder>/C:/runtime-
EclipseApplication//DEMOPROJECT</tns:projectFolder>
    <tns:ontologyURL>http://www.ict-peces.eu/ont/smartspace.owl</tns:ontologyURL>
    <tns:ontologyURL>http://www.ict-peces.eu/ont/device.owl</tns:ontologyURL>
  </tns:general>
  <tns:deviceInstances>
    <tns:contextRDF>project.owl</tns:contextRDF>
    <tns:device>
      <tns:id>0</tns:id>
      <tns:name>GUIDESYSTEM</tns:name>
      <tns:Type>Laptop</tns:Type>
      <tns:LocX>324</tns:LocX>
      <tns:LocY>185</tns:LocY>
      <tns:coordinatorRole>true</tns:coordinatorRole>
      <tns:gatewayRole>>false</tns:gatewayRole>

      <tns:CommunicationMethod>EmulationTransceiver</tns:CommunicationMethod>
      <tns:DeviceSecurity>
        <tns:CertName>certificate-unewdemo</tns:CertName>
        <tns:TrustLevel>full</tns:TrustLevel>
      </tns:DeviceSecurity>
      <tns:service>
        <tns:name>GuideService</tns:name>
        <tns:scope>Space</tns:scope>
      </tns:service>
      <tns:roleSpecificationScope>Space</tns:roleSpecificationScope>
      <tns:roleSpecification>
        <tns:id>TRADEGUIDE.pqry</tns:id>
        <tns:roleSpecificationTrustLevel>Don't
apply</tns:roleSpecificationTrustLevel>
      </tns:roleSpecification>
    </tns:device>
    <tns:device>
      <tns:id>1</tns:id>
      <tns:name>LOCATIONSYSTEM</tns:name>
      <tns:Type>Laptop</tns:Type>
      <tns:LocX>161</tns:LocX>
      <tns:LocY>182</tns:LocY>
      <tns:coordinatorRole>>false</tns:coordinatorRole>
      <tns:gatewayRole>true</tns:gatewayRole>

      <tns:CommunicationMethod>EmulationTransceiver</tns:CommunicationMethod>
      <tns:DeviceSecurity>
        <tns:CertName>certificate-unewdemo</tns:CertName>
        <tns:TrustLevel>full</tns:TrustLevel>
      </tns:DeviceSecurity>
      <tns:service>
        <tns:name>LocationService</tns:name>
        <tns:scope>Space</tns:scope>
      </tns:service>
    </tns:device>
    <tns:device>
      <tns:id>2</tns:id>
      <tns:name>VISITOR_IPAQ</tns:name>
```

```

        <tns:Type>IPAC614c</tns:Type>
        <tns:LocX>468</tns:LocX>
        <tns:LocY>112</tns:LocY>
        <tns:coordinatorRole>>false</tns:coordinatorRole>
        <tns:gatewayRole>>false</tns:gatewayRole>

    <tns:CommunicationMethod>EmulationTransceiver</tns:CommunicationMethod>
    <tns:DeviceSecurity>
        <tns:CertName>certificate-unewdemo</tns:CertName>
        <tns:TrustLevel>full</tns:TrustLevel>
    </tns:DeviceSecurity>
</tns:device>
<tns:device>
    <tns:id>3</tns:id>
    <tns:name>VISITOR_HTC</tns:name>
    <tns:Type>htc</tns:Type>
    <tns:LocX>460</tns:LocX>
    <tns:LocY>304</tns:LocY>
    <tns:coordinatorRole>>false</tns:coordinatorRole>
    <tns:gatewayRole>>false</tns:gatewayRole>

    <tns:CommunicationMethod>EmulationTransceiver</tns:CommunicationMethod>
    <tns:DeviceSecurity>
        <tns:CertName>certificate-unewdemo</tns:CertName>
        <tns:TrustLevel>full</tns:TrustLevel>
    </tns:DeviceSecurity>
</tns:device>
</tns:deviceInstances>
</tns:projectSettings>

```

Appendix 2 - project.owl

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://www.daml.org/services/owl-s/1.1/Service.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:j.2="http://www.ict-peces.eu/ont/device.owl#"
  xmlns:j.1="http://www.ict-peces.eu/ont/smartspace.owl#">
  <owl:Ontology rdf:about="http://www.example.com/project.owl">
    <owl:imports rdf:resource="http://www.ict-peces.eu/ont/smartspace.owl"/>
    <owl:imports rdf:resource="http://www.ict-peces.eu/ont/device.owl"/>
  </owl:Ontology>
  <j.1:SmartSpace rdf:about="http://www.ict-peces.eu/ont/smartspace.owl#BoothNavigation"/>
  <j.2:Member rdf:about="http://www.ict-peces.eu/ont/device.owl#VISITOR_IPAQ"/>
  <j.1:SmartSpace rdf:about="http://www.ict-peces.eu/ont/smartspace.owl#BoothMonitoring"/>
  <j.2:Gateway rdf:about="http://www.ict-peces.eu/ont/device.owl#LOCATIONSYSTEM">
    <j.0:provides>
      <j.0:Service rdf:about="http://www.daml.org/services/owl-s/1.1/Service.owl#LocationService"/>
    </j.0:provides>
  </j.2:Gateway>
  <j.2:Coordinator rdf:about="http://www.ict-peces.eu/ont/device.owl#GUIDESYSTEM">
    <j.0:provides>
      <j.0:Service rdf:about="http://www.daml.org/services/owl-s/1.1/Service.owl#GuideService"/>
    </j.0:provides>
  </j.2:Coordinator>
  <j.2:Member rdf:about="http://www.ict-peces.eu/ont/device.owl#VISITOR_HTC">
    <j.1:consumes rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl#GuideService"/>
  </j.2:Member>
  <j.1:SmartSpace rdf:about="http://www.ict-peces.eu/ont/smartspace.owl#TaxiBooking"/>
</rdf:RDF>
```

Appendix 3 - GuideSYSTEMContext.pctx

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://www.daml.org/services/owl-s/1.1/Service.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:j.2="http://www.ict-peces.eu/ont/device.owl#"
  xmlns:j.1="http://www.ict-peces.eu/ont/smartspace.owl#">
  <owl:Class rdf:about="http://www.ict-peces.eu/ont/device.owl#Coordinator"/>
  <j.2:Coordinator rdf:about="http://www.ict-peces.eu/ont/device.owl#GUIDESYSTEM">
    <j.0:provides rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl#GuideService"/>
    <j.1:consumes rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl#LocationService"/>
  </j.2:Coordinator>
</rdf:RDF>
```


Appendix 4 - events.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:simulationEventList xmlns:tns="http://www.example.org/project"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
diagramName="new_diagram.peces.eventdiagram">
  <tns:simulationEvent>
    <tns:name>LOCATIONSYSTEM_ON</tns:name>
    <tns:description>This is a Peces event</tns:description>
    <tns:type>deviceSwitchON</tns:type>
    <tns:delay>2000</tns:delay>
    <tns:contributingDevice>
      <tns:id>1</tns:id>
      <tns:name>LOCATIONSYSTEM</tns:name>
      <tns:URI />
    </tns:contributingDevice>
  </tns:simulationEvent>
  <tns:simulationEvent>
    <tns:name>VISITOR_HTC_ON</tns:name>
    <tns:description>This is a Peces event</tns:description>
    <tns:type>deviceSwitchON</tns:type>
    <tns:delay>5000</tns:delay>
    <tns:contributingDevice>
      <tns:id>3</tns:id>
      <tns:name>VISITOR_HTC</tns:name>
      <tns:URI />
    </tns:contributingDevice>
  </tns:simulationEvent>
  <tns:simulationEvent>
    <tns:name>VISITOR_IPAQ_ON</tns:name>
    <tns:description>This is a Peces event</tns:description>
    <tns:type>deviceSwitchON</tns:type>
    <tns:delay>5000</tns:delay>
    <tns:contributingDevice>
      <tns:id>2</tns:id>
      <tns:name>VISITOR_IPAQ</tns:name>
      <tns:URI />
    </tns:contributingDevice>
  </tns:simulationEvent>
  <tns:simulationEvent>
    <tns:name>GUIDESYSTEM_ON</tns:name>
    <tns:description>This is a Peces event</tns:description>
    <tns:type>deviceSwitchON</tns:type>
    <tns:delay>10000</tns:delay>
    <tns:contributingDevice>
      <tns:id>0</tns:id>
      <tns:name>GUIDESYSTEM</tns:name>
      <tns:URI />
    </tns:contributingDevice>
  </tns:simulationEvent>
  <tns:simulationEvent>
    <tns:name>DevConnection</tns:name>
    <tns:description>This is a Peces event</tns:description>
    <tns:type>connectionLinkChange</tns:type>
    <tns:delay>15000</tns:delay>
    <tns:contributingDevice />
    <tns:connectionLinkChanges>
      <tns:connectionLinkChange>
        <tns:source>
          <tns:id>0</tns:id>
        </tns:source>
      </tns:connectionLinkChange>
    </tns:connectionLinkChanges>
  </tns:simulationEvent>
</tns:simulationEventList>
```

```

        <tns:name>GUIDESYSTEM</tns:name>
      </tns:source>
      <tns:target>
        <tns:id>1</tns:id>
        <tns:name>LOCATIONSYSTEM</tns:name>
      </tns:target>
      <tns:type>connect</tns:type>
    </tns:connectionLinkChange>
  </tns:connectionLinkChange>
  <tns:source>
    <tns:id>0</tns:id>
    <tns:name>GUIDESYSTEM</tns:name>
  </tns:source>
  <tns:target>
    <tns:id>3</tns:id>
    <tns:name>VISITOR_HTC</tns:name>
  </tns:target>
  <tns:type>connect</tns:type>
</tns:connectionLinkChange>
<tns:connectionLinkChange>
  <tns:source>
    <tns:id>0</tns:id>
    <tns:name>GUIDESYSTEM</tns:name>
  </tns:source>
  <tns:target>
    <tns:id>2</tns:id>
    <tns:name>VISITOR_IPAQ</tns:name>
  </tns:target>
  <tns:type>connect</tns:type>
</tns:connectionLinkChange>
</tns:connectionLinkChanges>
</tns:simulationEvent>
<tns:simulationEvent>
  <tns:name>context_change</tns:name>
  <tns:description>This is a Peces event</tns:description>
  <tns:type>deviceContextChange</tns:type>
  <tns:delay>10000</tns:delay>
  <tns:contributingDevice>
    <tns:id>1</tns:id>
    <tns:name>LOCATIONSYSTEM</tns:name>
    <tns:URI>http://www.ict-
peces.eu/ont/device.owl#LOCATIONSYSTEM</tns:URI>
  </tns:contributingDevice>
  <tns:contextChanges>
    <tns:contextChange>
      <tns:instance>LocationService</tns:instance>
      <tns:URI>http://www.daml.org/services/owl-
s/1.1/Service.owl#LocationService</tns:URI>
      <tns:property>provides</tns:property>
      <tns:action>delete</tns:action>
    </tns:contextChange>
  </tns:contextChanges>
</tns:simulationEvent>
<tns:simulationEvent>
  <tns:name>DevDisconnection</tns:name>
  <tns:description>This is a Peces event</tns:description>
  <tns:type>connectionLinkChange</tns:type>
  <tns:delay>10000</tns:delay>
  <tns:contributingDevice />
  <tns:connectionLinkChanges>
    <tns:connectionLinkChange>
      <tns:source>


```


```

        <tns:id>0</tns:id>
        <tns:name>GUIDESYSTEM</tns:name>
    </tns:source>
    <tns:target>
        <tns:id>1</tns:id>
        <tns:name>LOCATIONSYSTEM</tns:name>
    </tns:target>
    <tns:type>disconnect</tns:type>
</tns:connectionLinkChange>
<tns:connectionLinkChange>
    <tns:source>
        <tns:id>0</tns:id>
        <tns:name>GUIDESYSTEM</tns:name>
    </tns:source>
    <tns:target>
        <tns:id>2</tns:id>
        <tns:name>VISITOR_IPAQ</tns:name>
    </tns:target>
    <tns:type>disconnect</tns:type>
</tns:connectionLinkChange>
</tns:connectionLinkChanges>
</tns:simulationEvent>
<tns:simulationEvent>
    <tns:name>Devconnection_new</tns:name>
    <tns:description>This is a Peces event</tns:description>
    <tns:type>connectionLinkChange</tns:type>
    <tns:delay>10000</tns:delay>
    <tns:contributingDevice />
    <tns:connectionLinkChanges>
        <tns:connectionLinkChange>
            <tns:source>
                <tns:id>0</tns:id>
                <tns:name>GUIDESYSTEM</tns:name>
            </tns:source>
            <tns:target>
                <tns:id>2</tns:id>
                <tns:name>VISITOR_IPAQ</tns:name>
            </tns:target>
            <tns:type>connect</tns:type>
        </tns:connectionLinkChange>
    </tns:connectionLinkChanges>
</tns:simulationEvent>
</tns:simulationEventList>

```

Appendix 5 - Questionary

	<p>PECES USER ACCEPTANCE QUESTIONNAIRE FOR THE EVALUATION OF THE DEVELOPMENT TOOLS</p>
<p>Part I – User Information</p>	
<p>1. Are you from industry or academia?</p> <ol style="list-style-type: none"> 1. Industry 2. Academic 	
<p>2. Are you member of PECES Interest Group?</p> <ol style="list-style-type: none"> 1. Yes 2. No 	
<p>3. Would you like to register to become a member of the PECES Interest Group?</p> <ol style="list-style-type: none"> 1. Yes 2. No <p>If your answer is Yes please provide your email address to be added to the PECES Interest Group mailing list</p> <p>-----</p>	

	<p>PECES USER ACCEPTANCE QUESTIONNAIRE FOR THE EVALUATION OF THE DEVELOPMENT TOOLS</p>
<p>Part II – User Acceptance Questionnaire</p>	
<p>1. On a scale from 1 to 5 please indicate how difficult is to develop PECES middleware application without the PECES Development Tools</p> <ol style="list-style-type: none"> 1. Very Difficult 2. Difficult 3. Neither Difficult Nor Easy 	

- 4. Easy
- 5. Very Easy

If your answer is very easy/ easy please explain what are the major factors forming your opinion

2. On a scale from 1 to 5 please indicate what is the general impression you have for the PECES Development Tools

- 1. Very Impressive
- 2. Impressive
- 3. Neither Impressive Nor Unimpressive
- 4. Unimpressive
- 5. Very Unimpressive

If your answer is very unimpressive / unimpressive please explain what are the major problems forming your opinion.

3. On a scale from 1 to 5 please indicate the level of training is required for the user to develop and test applications using the PECES Development Tools

- 1. Very Low
- 2. Low
- 3. Neither Low Nor High
- 4. High
- 5. Very High

If your answer is very high / high please explain what are the major problems forming your opinion.

4. On a scale from 1 to 5 please indicate how reliable is the PECES Development Tools for the middleware application development

- 1. Very Reliable
- 2. Reliable
- 3. Neither Reliable Nor Unreliable
- 4. Unreliable
- 5. Very Unreliable

If your answer is very unreliable / unreliable please explain what are the major problems forming your opinion.

5. On a scale from 1 to 5 please indicate how easy is to use the PECES Device Definition tool for the middleware application development

- 1. Very Easy
- 2. Easy
- 3. Neither Easy Nor Difficult
- 4. Difficult
- 5. Very Difficult

If your answer is very difficult / difficult please explain what are the major problems forming your opinion.

6. On a scale from 1 to 5 please indicate how easy is to use the PECES Ontology Instantiation tool for the middleware application development

- 1. Very Easy
- 2. Easy
- 3. Neither Easy Nor Difficult
- 4. Difficult
- 5. Very Difficult

If your answer is very difficult / difficult please explain what are the major problems forming your opinion.

7. On a scale from 1 to 5 please indicate how easy is to use the PECES Service Definition tool for the middleware application development

- 1. Very Easy
- 2. Easy
- 3. Neither Easy Nor Difficult
- 4. Difficult
- 5. Very Difficult

If your answer is very difficult / difficult please explain what are the major problems forming your opinion.

8. On a scale from 1 to 5 please indicate how easy is to use the PECES Role Specification tool for the middleware application development

1. Very Easy
2. Easy
3. Neither Easy Nor Difficult
4. Difficult
5. Very Difficult

If your answer is very difficult / difficult please explain what are the major problems forming your opinion.

9. On a scale from 1 to 5 please indicate how easy is to use the PECES Event Definition tool and PECES Event Diagram tool for modelling smart space networks dynamics

1. Very Easy
2. Easy
3. Neither Easy Nor Difficult
4. Difficult
5. Very Difficult

If your answer is very difficult / difficult please explain what are the major problems forming your opinion.

10. On a scale from 1 to 5 please indicate how easy is to use the PECES Testing tool for simulating (testing) smart space networks application.

1. Very Easy
2. Easy
3. Neither Easy Nor Difficult
4. Difficult
5. Very Difficult

If your answer is very difficult / difficult please explain what are the major problems forming your opinion.

11. Do you think that the PECES Development Tools are very useful for middleware application development and testing?

- 1. Yes
- 2. No

If your answer is **Yes** please explain what are the major features forming your opinion.

If your answer is **No** please explain what are the major problems forming your opinion.

12. Please list the ways in which the PECES Development Tools could be improved

13. How do the PECES Development Tools compare with other tools you have used to develop similar middleware applications

14. Do you have any further comments relating to the usability of the PECES Development Tools

