

Model-driven and Compositional Service Creation in the Internet of Services

Selo Sulistyo

**Model-driven and Compositional Service
Creation in the Internet of Services**

Doctoral Dissertation for the Degree *Philosophiae Doctor (PhD)* in
Information and Communication Technology

University of Agder
Faculty of Engineering and Science
2012

Doctoral Dissertation by the University of Agder 45

ISBN: 978-82-7117-708-9

ISSN: 1504-9272

©Selo Sulisty, 2012

Printed in the Printing Office, University of Agder
Kristiansand

Preface

This work was supported by the Norwegian Research Council (NFR) in the context of the ISIS project that was done during the period of 2007-2011. I would like to convey my sincere thanks to Reidar Martin Svendsen from Telenor ASA, the leader of the ISIS project, for the things that made possible to run the research work.

Many other people have helped me through the course for the pursuing of my doctoral degree. First of all, I would like to give my sincere thanks to my supervisor, Professor Andreas Prinz and co-supervisor Professor Frank Reichert. Without their guidance and help, the completion of the thesis would have been a mission impossible. I benefited from their rigorous style of work and their strict requirements on the quality of research. Professor Andreas Prinz's inspirations through discussions and valuable feedbacks made the journey of research more effective and enjoyable. The art of research and the skills of writing scientific papers have also been accumulated through the journey.

I would also like to convey my sincere thanks to my parents for their unselfish love and support for all the years, to my dearest wife Yunita Widayastuti for her continuous supports and love, to our lovely children, Hanif, Irfan, Dania, and Safira for all their smiles that can dismiss all the troubles and tiredness, to all my friends and colleagues in UiA, Jan Pettersen Nytnun, Terje Gjoesaeter, Liping Mu, Xuan, Xin He, Yi Ren, and Jason for the very nice discussions, and last but not least, to my colleagues in Department of Electrical Engineering Gadjah Mada University Indonesia for letting me several years off.

Selo Sulistyono
Grimstad, September 2012

Abstract

In the Future Internet, billions of devices will be connected to the Internet. Devices at any levels of hierarchy provide software functionality that can be used by others. We can call the device's functionality a service, which in turn, introduces the concept of the Internet of Services. From the software developer perspectives, a new service can be created by utilizing services in the Internet of Services. An important issue of the creation of such service-based application is regarding their deployment method on personalized and embedded devices. For each device with different capability and configuration, different tailored code is required. For this, a flexible method and tools that support an automatic code generation for a device with a specific capability and configuration are mandatory.

This thesis proposes PMG-pro (Present, Model, Generate and provide), a language-independent, bottom-up and model-driven method for the service creation in the Internet of Services. With this method, a service is created by providing the new functionality of a service-based application as a service. By using existing service frameworks and APIs, from a service description, PMG-pro generates an abstract graphical service representation (service model) and source code implementing for service invocations. Depending on the target modeling languages, different graphical notations can be used to represent services. Similarly, different programming languages can also be used to implement the service invocations. We call these pairs (i.e., the service model and the source code) platform-specific models. With these platform models, service composers use the graphical service representation to model new service-based applications, while the machine (i.e., computer system) uses the source code to generate code from the service-based application model.

This thesis contributes to the service engineering method that applies a model-driven development approach. Three main contributions are a model-driven method for service creation, an automatic service presentation of pre-made services, and a new method of handling device capability and configuration. With these, service creation in the Internet of Services can be done in a rapid and automatic manner. Service designers can create a new service by defining a model of service-based applications using pre-made service models, while code for a specific device can be generated automatically from the model.

The PMG-pro method has been partly prototyped and validated on various case studies in the domain of smart homes that have produced encouraging results. The method promotes a rapid, language-independent, and unified process of software service development.

Contents

List of Figures	xv
List of Tables	xvii
Definitions and Abbreviations	xix
1 Introduction	1
1.1 An Introduction to the Area of Interest: <i>the Future Internet</i>	1
1.2 Motivation	6
1.3 Main Contributions	8
1.4 Thesis Structure	11
2 Software Development in the Internet of Services	13
2.1 Service-based Applications	13
2.2 Challenges and Problems	16
2.3 Further Research Questions	24
3 Background	29
3.1 Service-oriented Systems	29
3.2 Model-driven Development	44
3.3 Embedded Systems	52
3.4 Software Product Lines	55
3.5 The ISIS Project	56
4 The PMG-pro Method	59
4.1 Overview of PMG-pro	59
4.2 The Service Presentation and Abstraction Step	60
4.3 The Service Library and the Service Frameworks (APIs) Repository	72
4.4 The Modeling Step	74
4.5 The Code Generation and Providing Step	78

5	Proof-of-Concept	83
5.1	Overview	83
5.2	Service Presenter	84
5.3	The Service Library	91
5.4	Code Generator	92
5.5	Case Studies	93
6	Evaluation	113
6.1	Evaluation of the Method and Case Studies	113
6.2	Evaluation against Research Questions	118
6.3	Evaluation against Related Work	120
7	Summary and Future Work	131
7.1	Summary	131
7.2	Future Work	133
	REFERENCES	137
A	List of Publications	149
B	UPnP Light Service Description	151
C	XMI Representation of ARCTIS Building Blok of UPnP Light	155
D	Code for Invocation of UPnP Light services	161
E	Presenting Embedded Services for End-user Composition	165
	INDEX	170

List of Figures

1.1	The Service Engineering Domain [11]	6
2.1	Software-based Internet services (SBIS) and Software-enabled Internet Services (SEIS). SEIS may use SBIS.	14
2.2	A conceptual model of service-based applications. New services can be created by providing the new composite functionalities (composite service) as a new service.	15
2.3	Different actors have different roles in the development and use of service-based applications. To some extent, end-users are a type of service composers.	17
2.4	A conceptual model of a service presented in [47]. In this model, a service system may be developed by composing service components, since service components provide services.	19
2.5	A conceptual model of a service presented in [103]. A service has a service description. A service performs operations.	20
3.1	The historical perspective of the use of different models of a software unit with regard to the abstraction levels of software unit. A service is seen as a new model of a software unit.	31
3.2	An example of a service presentation using UML classes.	33
3.3	An example of service presentation using a SCA component. SCA describes events, operation, and parameters. This promotes both request-reply based and event-based applications modeling. The figure was taken from [40].	34
3.4	An example of service presentation using a SoaML participant. The picture is taken from [79].	35
3.5	A conceptual model of UPnP devices presented in a UML class diagram. A UPnP device has two different descriptions: device and service descriptions.	37

3.6	A meta-model of Web service description presented in a UML class diagram.	38
3.7	An example of Web service description. The names of the services (operations) are listed in the description.	39
3.8	An example generated classes from WSDL2Java in the AXIS web service framework. The classes are used to invoke the actual services that reside on a service provider.	40
3.9	An example of OSGi service description presented in XML. The actual service description can only be seen at run-time.	42
3.10	OMG's Model-driven architectures (MDA). Different actors have different roles in the development of software systems. The idea is to separate the problem with the solution.	45
3.11	An illustration picture showing the relation between systems, models and languages.	48
3.12	A conceptual model of model transformation.	49
3.13	A concept of code generation process in the domain-specific modeling languages (DSM). The is a strong relation between the modeling language, the domain framework and code generation.	51
3.14	A basic mechanism of information exchange between a UPnP device (Media Renderer) and a UPnP Control Point (mobile phone) - Retrieving Device and Service Descriptions	53
3.15	The use of device identification for service customization. Different devices need tailored code with respect to its capabilities and configurations.	56
3.16	The service-driven development process that has been used in the ISIS project.	58
4.1	The PMG-pro architecture. It consists of the presentation and abstraction step, the modeling step, and the code generation and providing step.	60
4.2	The service presenter and abstractor of the PMG-pro. The presents and abstracts services so that service integrators can compose them in a model-driven environment.	61
4.3	A simplified model of service descriptions in the PMG-pro. The model is adapted from the conceptual model of services presented in [103] and [47]. All service properties are described in the service description.	62

4.4	The relation between a service description, its model (graphical representation) and source code. The concrete service itself is available at run-time. Services users do not know how the service has been implemented.	64
4.5	A simple building block model to represent a UPnP device (and its service). A UPnP device has inputs (UPnP actions) and generates events (UPnP state variables). A meta-model of UPnP devices is presented in Figure 3.5	65
4.6	A conceptual model of ARCTIS building block - A basic element of model in ARCTIS [53, 57, 65].	66
4.7	An ARCTIS building block is used to visually represent the UPnP Light service	68
4.8	A SCA component is used to represent the UPnP light service. . . .	68
4.9	UML classes are used to represent the UPnP Light service.	69
4.10	A SoaML participant is used to represent UPnP Light service. . . .	69
4.11	A platform independent service models and its code. In this example, a Lamp is a common model of different implementation of services: UPnP_Lamp, DPWS_Lamp and UPnP_Light services. . . .	70
4.12	An example of taxonomy of service models. Each model is connected to source code. The source code itself can be at any programming language.	73
4.13	A General formal ontology of smart homes devices used in PMG-pro.	74
4.14	A service-based application model is specified using an activity diagram. The figure shows a service-based application that uses three existing services. Each service has internal activities. A swimlane is representing a service	75
4.15	A service-based application model specified is using a sequence diagram. In this figure, three different UPnP services are used to build a service-based application. The application will play music when a UPnP_Light service is invoked, and will stop it on a certain value of weather parameter.	76
4.16	The code generator. A template is used to generate source code. . .	78
4.17	A merge node. The figure shows an activities and three input data flows. The activity will be executed all the data flows are arrived. .	79
5.1	The screenshot of the user interface of PMG-pro.	83
5.2	An example of a UPnP service description. The actions are listed in the service description. See Appendix B for more detail.	85

5.3	The internal activity diagram of a simple ARCTIS building block for UPnP services. The building block has one action (operation) and one event out. This building block is used for the construction the template for a UPnP ARCTIS building block.	86
5.4	XML representation of an ARCTIS building block for the construction of the template for a UPnP ARCTIS building block.	87
5.5	XML representation of the UPnP Light service in an ARCTIS building block presented in Figure 4.7. The XML contains information about operation names, xmi:id, arguments, type of arguments, directions, etc.	89
5.6	Smart homes - Selected environment for the purpose of proof-of-concept. All devices provide services that can be invoked. We have three case studies based on this environment. Each case study has o scenario of service-based application.	93
5.7	An abstract representation of the UPnP display service in an UML class.	95
5.8	An UML sequence diagram of the alarm system in the scenario. . .	97
5.9	Events of the UPnP Light service on Intel's SDK for UPnP- <i>Device Spy</i>	99
5.10	An extending scenario of the UPnP network.	101
5.11	The feelgood model is presented in an ARCTIS building block diagram. Three building blocks are used to describe the combined application.	104
5.12	The ARCTIS model of the new application. The included service models specify the structure of the application, while the interactions between service models specify the behavior.	108
5.13	The new composite service at run-time. The new service-based application is provided as a new UPnP service.	111
6.1	The present, model, and generate-provide method for building service-based applications.	114
6.2	PMG-pro contribution to the ISIS method	124
C.1	An ARCTIS building block is used to represent run-time UPnP Light services, see also Figure 4.7	155
E.1	A simplified UML class diagram of an ICE edge. An ICE edge can be either an action or an trigger edge.	167
E.2	An ICE Puzzle represents UPnP MediaRenderer.	168

E.3 The models of the new application defined in the scenario. Two puzzle compositions specify the structure and the behavior of the application. 168

List of Tables

4.1	Transformation rules between UPnP - ARCTIS Building Block . . .	67
5.1	A simple database for implementing the Service Library for the purpose proof-of-concept.	91
5.2	Transformation rules between UPnP - UML class	95
5.3	A transformation of an UPnP service description into an UML class	96
5.4	A Transformation of an UPnP service description into an ARCTIS building block	103
6.1	Evaluation - A comparison	129
E.1	Transformation rules between UPnP - ICE Edge	167

Definitions and Abbreviations

BPEL	Business process execution language
Bundle	Applications or components that can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot.
CIM	Computational independent model
Composite services	Services that use other services.
Devices	Any software-based entity that is connected to the Internet.
DPWS	Device profile for web services
DSML	Domain-specific modeling language
DSL	Domain-specific language
Edge	An bundle that provides service functionality in the form of triggers and actions.
Embedded services	Services that embed on devices
Embedded devices	Any microprocessor-based entities that have input, output and process. It runs a program. Examples are copy machines, washing machines and mobile phones.
End user	A person who uses services
ISIS	Infrastructure for Integrated Services
MDD	Model-driven development
MDA	Model-driven architectures
Mobile devices	Any portable microprocessor-based systems with regard to the Internet connectivity. It can be a laptop, mobile phones, etc.
OSGi	Open Service Gateway initiative.
OSGi framework	A module system and service platform for the Java programming language that implements a complete and dynamic component model.
Personalized devices	Any configurable device according to the users
PIM	Platform-independent model
PSM	Platform-specific model
PMG-pro	Present, Model, Generate and provide
SBA	Service-based applications
SBIS	Service-based Internet Services
SCA	Service component architectures
SDP	Simple description protocol

SdS	Software-driven Services
SEIS	Service-enabled Internet Services
Service architect	A person who specifies services, both simple and composite services.
Service composer	A person who composes services: end-users or service integrators.
Service developer	A person who implements basic services
Service description	A description of the properties of a service. It contains information about the services names, end points, etc.
Service integrator	A type of software developers that create service by composing services
Service provider	A type of organization that runs and provides services for service users
Service user	A person or an organization or a device or an application who use the services.
Simple services	This is a relative term. They can be composed to create service-based applications.
Smart homes	It is automation of the home, housework or household activity. It may include centralized control of lighting, HVAC, appliances, and other systems, to provide improved convenience, comfort, energy efficiency and security.
SOA	Service-oriented architecture
SoaML	Service-oriented Modeling Language - an UML profile
Software Developer	A person or organization concerned with facets of the software development process.
SSBS	Software and Software-based Services
SSDP	Simple service description protocol
UML	Unified modeling language
UPnP	Universal Plug and Play. An example of device coordination framework.
XML	eXtensible Markup Language
WSDL	Web service description languages
WS-BPEL	Web service - business process execution language

Chapter 1

Introduction

This introductory chapter aims to give an overview of the thesis context, starting with an introduction to the area of interest in Section 1.1, which focuses on the Future Internet and software service engineering. This includes an introduction of the definitions of a service, a service-based application and a composite service. The aim is showing the position of software engineering in the Future Internet. A presentation of the motivations for the thesis follows in Section 1.2, and the main contributions of the thesis to the field of software service engineering are presented briefly in Section 1.3. At the end of this chapter, in Section 1.4, the structure of the thesis is outlined.

1.1 An Introduction to the Area of Interest: *the Future Internet*

It is undeniable that the Internet is becoming an important infrastructure for the growth of today's economy and society, where the infrastructure itself is also gradually becoming larger. Now, information technology is spreading into all areas of daily life, leading to an increasing amount of information and applications. New devices and technologies (e.g., protocol layers) are continually added, leading to an increasing complexity of software systems. Surprisingly, today's Internet that was designed in 1970s is mainly built only for information sharing. There is a mismatch between the original design goals and the current and future utilization of the Internet technology. A new system is needed, and we call this system the Future Internet [83].

In the Future Internet, Internet resources (people, media, services, devices, and networks) will be converged in terms of connectivity. Various autonomous comput-

ing and networked devices, from small (mobile devices, embedded systems, etc.) to powerful devices (desktops and servers) may be easily connected to the Internet network, in a plug-and-play manner. These devices can communicate and cooperate with each other to form a specific composite system or application. So, the cooperation is not only between people and devices, but also between devices. This is supported by the fact that as on-going miniaturization allows small devices such as sensors and actuators, to become autonomous, smart and powerful data-processing engines of all kinds are emerging. In contrast, the more devices are connected to the Internet the more applications are going to be diverged, leading to an increasing complexity of software systems and their development.

1.1.1 The Concepts of the Future Internet

To illustrate the Future Internet, several terms and related concepts have been introduced. In this section, three concepts are presented: the Internet of Things [105], the Internet of Services [100] and the Cloud Computing [114]. We will show how the concepts are related to each other.

The first important concept illustrating the Future Internet is the Internet of Things [105]. Different perspectives may have different definitions of the *things* in the Internet of Things. However, in the context of the Internet technology, the thing can be defined as a physical or virtual entity that exists in space and time, and is capable of being identified and utilized according to its properties. Among important properties are the ID, location, name and behavior.

In [105] it is stated that:

The Internet of Things allows people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network and any service.

The definition above implies handling of several important issues. Two examples of the issues are how the connection between entities can be made seamlessly, and which addressing mechanisms should be used. Obviously, in order to be able to communicate with each other, each device requires a different address, an ID and a name that must be unique.

Referring to the object-oriented perspective, the Internet of Things can also be considered as the Internet of objects. The objects would be equipped with sensors enabling them to detect the environment and to adapt if the environment is changed. In a smart home environment [86, 22, 109] for example, all objects (devices) are

considered to be equipped with minuscule identifying devices and they are connected to the network. They are a kind of autonomous and smart objects that are able to do self-adaptation to the network. It will be possible to develop combined applications based on individual object's behaviors.

The objects in the Internet of Things provide or/and use functionalities. We can call the object's functionalities services, which in turn, introduces the term Internet of Services [100]. The vision of the Internet of Services is that everything is available on the Internet as a service, such as the software itself, the tools to develop the software, and the platform (operating systems, hardware and networks) to run the software. This is the main characteristic of the concept of the Internet of Services.

Another important concept of illustrating the Future Internet is Cloud computing [114]. The concept is a relatively new model of Internet-based computing, where resources in the Internet, e.g., servers, storage, networking, software, are provided as services. Cloud computing can be defined as an architecture where IT resources (hardware and software systems) are delivered to the users on demand without strict dedicated associations between customers and resources [114]. The delivery model for the resources is based on the utility using a pay-per-use pricing model. The key is that any kind of user can access the services, even inexperienced users. The main feature of all these service offering models is that they break up the previously monolithic ownership and control of the resources in the various technology layers and distribute them across multiple entities.

The concept of Cloud computing has a similarity with the concept of the Internet of Services, in which everything is provided as a service. Based on target service users, services in Cloud computing can be roughly divided into three main categories: infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS).

In the IaaS category, machines are virtualized and provided as services. These services include computational resources such as hardware and storage services including services for managing groups of these resources. The PaaS service category offers development platforms for various application domains. In the SaaS category, software applications are provided as services. The users of these services range from private users of services to enterprise customers operating entire business processes.

The Cloud computing concept has also a similarity with the concept of the Internet of Things. Cloud computing uses an implicit assumption that resources are of a type of entities that is able to host and process data. However, this is the main issue in the Internet of Things. In the Internet of Things, every device is a networked,

smart and self-adapted resource. The devices also host and process data.

A common similarity of the three concepts mentioned above is about functionality (i.e., service) that is provided by an entity and can be used by others. The literature provides many informal definitions of a service that were inspired mainly by applications in the telecommunications domain. However, generally, a service can be defined as functionality offered to a service user by a service provider. Both service users and service providers can be human beings, enterprises, as well as software and hardware entities (i.e. programs and devices). In the context of the Internet of Things, services are software systems that are embedded on devices. However, it must be noted that services are not only hosted on small and embedded devices, but could also be hosted by more powerful devices, physically or virtually. With this situation, the Internet will contain a huge amount of embedded services.

The problem is that there is no standard of formal method and language for implementing services. The services may have been implemented in different programming languages. They can also run on different unrelated environments. This leads to a problem when the service users want to use the services in a composite manner. To solve the problem, a service broker in terms of service-oriented architecture (SOA) [28] can be considered. However, SOA is only a concept. We need tools and methods for implementing the concept. A rapid, flexible, high abstraction level and efficient tools for the development software service is required.

1.1.2 The Software Engineering Perspective

The classic issue of software development concerns the development methodology and languages (i.e. programming and modeling languages). There is still criticism of software systems development languages and the methods employed, such as high cost, long time-to-market, and poor flexibility. Software reuse is one of the proposed solutions to the criticized development methodology as it promotes reductions in cost and time-to-market. In this solution, a new software system can be promoted (created) by means of collaboration of existing software units. An example of this solution is the use of software components (i.e., software unit) instead of manually handwritten code from scratch to develop a software system, that was proposed by McIlroy in [62]. Based on his idea, several models of a software unit were introduced. Modules, objects, components are examples of models of a software unit.

However, the demand for software to live in an open world and to evolve continuously as the world evolves, is leading to the evolution of software methodologies and technologies. The evolution can be seen as a progressive journey from rigid

to flexible, static to dynamic, centralized to distributed solutions [25]. The history of software engineering shows a progressive departure from the strict boundaries of the closed-world assumption toward more flexibility to support continuous evolution [4]. Over the past years a major step of evolution in this direction has been made possible by the birth of the concept of a service.

With regard to the software component models introduced in [62], we consider that a service is only another type of model of a software unit. In this context, a service has evolved considerably from the older models: modules, objects, and components. They are only different in terms of abstractions, encapsulations and ownerships. Among these models, the service is the highest abstraction level and the most encapsulated model. Accordingly, depending on the models of software units, software systems can be categorized as module-oriented systems, object-oriented systems, component-oriented systems and service-oriented systems. The service-oriented system is also known as a service-based application.

Unfortunately, traditional software engineering methods and approaches are not fully appropriate for the development of service-based applications. In the context of service-oriented systems, these limitations have led to the emergence of software service engineering (SSE) as a new specialist discipline. However, research activities in this area are still immature, and many open issues remain. There is an urgent need for the research community and industry practitioners to develop comprehensive engineering principles, methodologies and tools support for the entire software development lifecycle of service-based applications [106].

Service engineering can be defined as the set of methods, techniques and tools to specify, design, implement, verify, and validate services that meet user needs and deploy and exploit these services, over current or future networks [116]. For the service development, service developers must be able to specify services so that they can be discovered and used and must be able to test these services in such a way that the tests reflect their complex and heterogeneous operating environment. According to [116], service engineering covers three important domains:

1. Service creation and tools: A software engineering platform.
2. Service management: The way a service is run and accessed by the users.
3. Network architecture: The way a service is delivered to the users.

Similarly, in [11], see Figure 1.1, the first domain is introduced as service engineering domain, the second domain is called service deployment and execution domain, and the third domain is called a service platform. The service engineering domain includes methods and tools (i.e., service creation environment, SCE) for

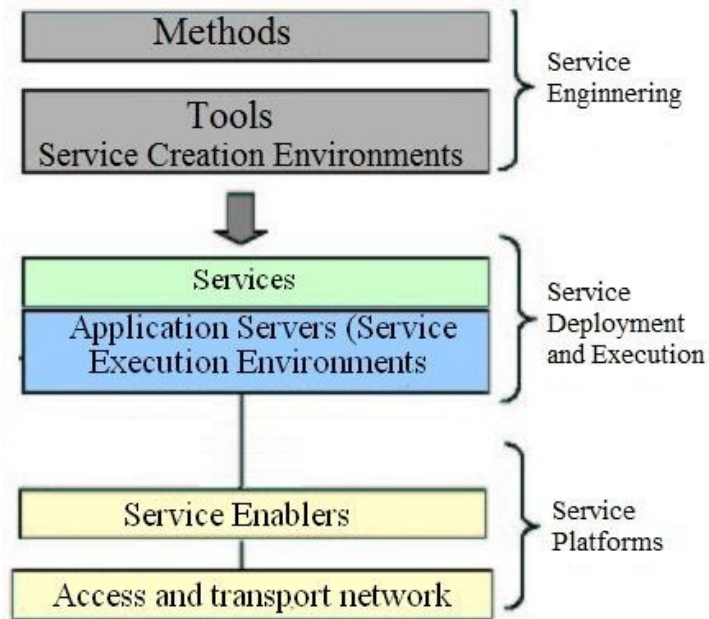


Figure 1.1: The Service Engineering Domain [11]

software services. This thesis addresses on the first domain of service engineering [116], that is service creation method and tools [[11]].

1.2 Motivation

Only with appropriate software will it be possible that the Future Internet comes to life as imagined. Within the context of the Internet of Services, creating service-based applications using the huge amount of services is a challenge. This includes the use of resources that are delivered in form of IaaS, PaaS and SaaS models in the context of Cloud computing. Within the European community [1], software development by composing pre-made services has been proposed as one of the research agendas during the years 2010-2015.

For the creation of service-based applications, different approaches and methods may be used. However, the approaches and methods should meet the three criteria below.

1. They must support for *exploring the reusability* of existing services.
2. The design processes should not depend on the target environment.
3. The service developers and service integrators should be shielded from complexity.

Two examples of approaches and methods are software composition approaches [3] (was originally introduced in [62]) and model-driven development approaches (MDD) [76]. The first approach supports exploring the reusability of software components. In the last decade, the approach has been adopted in several of today's large-scale software projects in the area of distributed systems. With regard to the software component that was introduced in [62], we treat a software component as a software unit. For the composition of software units, software composition systems [3] can be applied.

The second approach uses models to specify, design, implement, verify, and deploy software systems. When doing model-driven development, software developers will work on model levels in a platform independent manner. The reason is that the MDD focuses on the separation of the problem domain from the implementation domain. This enables software developers to focus on the problem solutions on a high abstraction level rather than the implementation details. By doing this, they do not need to pay attention to how the service models will be implemented considering the fact that the implementation is done automatically by machines (i.e., code will be generated automatically).

With regard to the complexity of software systems, the aims for both software composition [62] [3] and model-driven development (e.g. MDA [76]) are similar in which they are used for managing the complexity of software systems and their development. Having benefited from these approaches, the author of this thesis has been motivated to propose the use of MDD for the service creation in the Internet of Services. Employing this idea, the composition of services can be done using models at different abstraction levels, while the executable composite services can be generated automatically.

In fact, the use of models for the development of software applications has been done for a long time. For example, software developers often use high-level models (e.g. instance, box, and arrow sketches) to reason and communicate about a software system. The main problem with the use of high-level models is that they are almost always inaccurate when the design models would be implemented using specific programming languages manually [68]. This is a serious problem, since a model in MDD is used as a formal specification of a software system. A software system should be generated automatically from this formal specification, either by model interpretation or code generation.

Services in the Internet of Services may be hosted by small devices. Of the small devices, this thesis focuses on personalized and embedded devices which have different capabilities and possible configurations. For the development of such em-

bedded services, concrete and low-level information about the device capability and configuration is often required at the beginning of the development processes. In contrast, within the MDD context, such low-level information will be given later at the lowest level of design models, which is usually done at the end of the development processes. This is because the MDD development process goes from abstract (i.e., models) to concrete (i.e., implementations). If we want to apply MDD approaches for the development of embedded services, then we have to abstract the concrete information about the devices onto model levels. This abstraction process goes from concrete to abstract. So, there is a mismatch between the MDD approach and the software development approach for personalized and embedded devices.

The fundamental research question of this thesis is formulated as follows:

To what extent and how - can model-driven development approaches be applied for the creation of composite services and their deployment on personalized and embedded devices in the Internet of services?

The problems and challenges of the use of MDD for the development of service-based applications will be further analyzed and presented in Chapter 2. Furthermore, the fundamental research question above will also be further decomposed, and analyzed and presented into six sub questions in Section 2.3.

1.3 Main Contributions

Model-driven development is not a new methodology for software engineering. However, the idea of applying the model-driven development methodology to service creation in the Internet of Services is relative new. This thesis focuses on service engineering as introduced in [11] which includes methods and service creation tools. More specifically, the thesis focuses on model-driven and compositional methods for service creation. The thesis proposes a method for composing services and for providing the composite functionality as a new service. The contributions of the thesis are listed as follows:

1. A New Service Engineering Method for Service Creation in the Internet of Services.

The main problem of service development in the Internet of Services is that the *things* in the Internet of Things might implement services using different technologies and might also use different programming languages. A methodology to develop service-based applications using services that have been

implemented using different technologies is needed. Furthermore, when the service-based application has been developed, questions about how the applications can be provided as a new service and how the services will be deployed in different devices with different capabilities and various device configurations are interesting to be solved. The thesis contribution is a methodology of service engineering that is independent of technologies and supports for exploring the reusability of existing services. The thesis proposes a method called PMG-pro (present/abstract, model, generate and provide) to answer the fundamental research question.

The method has been presented in the paper "PMG-pro: A model-driven method for the development of service-based applications in a heterogeneous services environment" [97] and is published in the *Proceeding of IEEE International Conference on Software Engineering and Service Sciences, ICSESS 2010* and the *Proceeding of SDL Conference on Software 2011* [98]. A poster "Model-driven Approaches for Service-based Applications Development" [96] of the method was presented in the *Proceeding of 5th International Conference on Software and Data Technologies, ICSOFT 2010*.

2. An Automated Service Presenter and Service Abstractor.

Abstraction is an important key for the success of applying MDD. A service model is an abstract representation of a concrete service. Using abstract service models, new service-based applications can be modeled on model levels. Thus, we need a service presenter to present services, and a service abstractor to abstract services.

The presentation process involves model transformation methods [78, 8, 75]. The service presenter transforms a service description of a concrete service at run-time, to an abstract graphical service model. Different notations or symbols that conform to the chosen modeling languages can be used to present the services. The service graphical representation is an abstract service model. A service abstractor is used to construct more abstract service models in hierarchical service taxonomy.

The thesis contribution is an automated service presenter that is able to present graphically (embedded-) services from service descriptions and a service abstractor that constructs more abstract models. With the service presenter and service abstractor, it would be possible to present concrete services as abstract services. Using the abstract services, software developers can model new service-based applications using different embedded software systems

on different devices, on the model level. The contribution of the service presenter has been presented in the paper "Presenting Reusable Service Models in Model-driven Service Engineering" [93] and is published in the *Proceeding of International Conference on Future Information Technology, ICFIT 2010*.

A special service presenter has been developed for supporting end-user service compositions at run-time. The contribution has been presented in the paper "An Automated Services Presentation Method for Supporting End-user Compositions" [91] and is published in the *Proceeding of Conference on Information Technology and Electrical Engineering, Yogyakarta, Indonesia*.

3. A new method of handling device capabilities and configurations.

The promise with the model-driven development approaches for the development of software applications is that new applications can be specified, designed, analyzed, and verified by models and thereby, code for specific platforms of target devices can be generated automatically from the models to have running applications. However, for small devices (mobile phones for example), device capabilities and configurations of the target devices are unknown at design-time. This introduces difficulties for the automated code generation. To be able to have a fully-automated code generation, we need to have knowledge of the underlying platforms.

In addition the graphical service model, the service presenter in PMG-pro also generates source code conforming to a specific programming language. The source code implements the service invocations to the concrete service. We call these pairs (i.e., the service model and the source code) platform models. The platform models are constructed in a hierarchical manner. Software developers can use the service models to create new service-based applications, while the machine (i.e., code generator) use the source code to generate tailored code for specific target device with a specific device capability and configuration. With this, device capabilities and configuration can be managed.

The method has been partly presented in the paper "PMG-pro: A model-driven method for the development of service-based applications" and is published in the *Proceeding of SDL Conference on Software 2011* [98].

1.4 Thesis Structure

The thesis consists of 7 chapters where Chapters 2 and 3 present background information and studies which aim to identify the methodology and supporting tools for the service creation in service engineering. Chapters 4 and 5- present the proposed methods and proof-of-concepts. Evaluations of the proposed method and a summary are presented in Chapters 6 and 7, respectively.

- Chapter 1, "Introduction" gives an introduction to the context of this thesis and the area of interest. This includes a brief identification of the main problems in the area and the motivations for the work. Based on the motivation, main contributions of the work are also outlined in this chapter.
- Chapter 2 "Software Development in the Internet of Services" gives further explanation of challenges and problems of the use of model-driven development for the service creation in the Internet of Services. This includes further decomposition and analysis of the fundamental research question presented in Section 1.2. The chapter also presents basic definitions used in the thesis.
- Chapter 3 "Background" gives on an overview of the theoretical background related to the software service development in the Internet of Services. It presents briefly a theoretical background of service-oriented systems, model-driven development, embedded systems and software product lines. A short description about the ISIS project is also presented in this chapter. This thesis was done in the context of the ISIS project.
- Chapter 4, "The PMG-pro Method" presents a model-driven development method of software service in the Internet of Services. It is started with an overview of the PMG-pro method. PMG-pro consists of four steps: the abstraction and presentation step, the modeling step, the code generation step and the providing step. These steps are presented in different subsections. The method shows how the theory from Chapter 3 can be used to answer the research questions.
- Chapter 5 "Proof-of-Concept" presents a prototype which implements the PMG-pro method presented in Chapter 4. It includes a presentation of three case studies of the development of service-based applications in smart homes environment.
- Chapter 6 "Evaluation" presents evaluations of the proposed method. Firstly,

a general evaluation is presented in the first section. Evaluations against the research questions and related work follow in the following sections.

- Chapter 7 "Summary and Future Work" is the concluding chapter where the proposed solution and contributions are summarized. The chapter also contains some suggestion for future work in the area of model-driven service development in the Internet of Services.

Chapter 2

Software Development in the Internet of Services

A methodology and tool for the creation of services in the context of Internet of Services are the focus of this thesis. This chapter presents more detailed challenges and problems of software development in the Internet of Services that have been presented in Chapter 1. Section 2.1 presents an overview of service-based applications. This includes also a short introduction of terms and definitions that are used in this thesis. Section 2.2 discusses the identified challenges and problems of creating services in the Internet of Services. From the discussed challenges and problems, the fundamental research question presented in Section 1.2 is further decomposed into seven sub research questions.

2.1 Service-based Applications

Over the last decade, the service sector has become the biggest and fastest-growing business sector in the world [90]. In order for this growth to continue, services should become more accessible and should also yield higher business productivity. The advanced use of information technology can significantly help to achieve this continuity. Many different companies and research institutes have started to explore different aspects of the service sector to determine which services can be managed through information technologies (IT). Here, we are talking about Software and Software-based Services (SSBS).

In the enterprise context, SSBS will play important roles of keeping continuity of business sector growth. According to [90], SSBS in the Internet of Services can be either Software-based Internet Services (SBIS) or Software-enabled Internet Services (SEIS). The differences between SBIS and SIES in terms of value generated

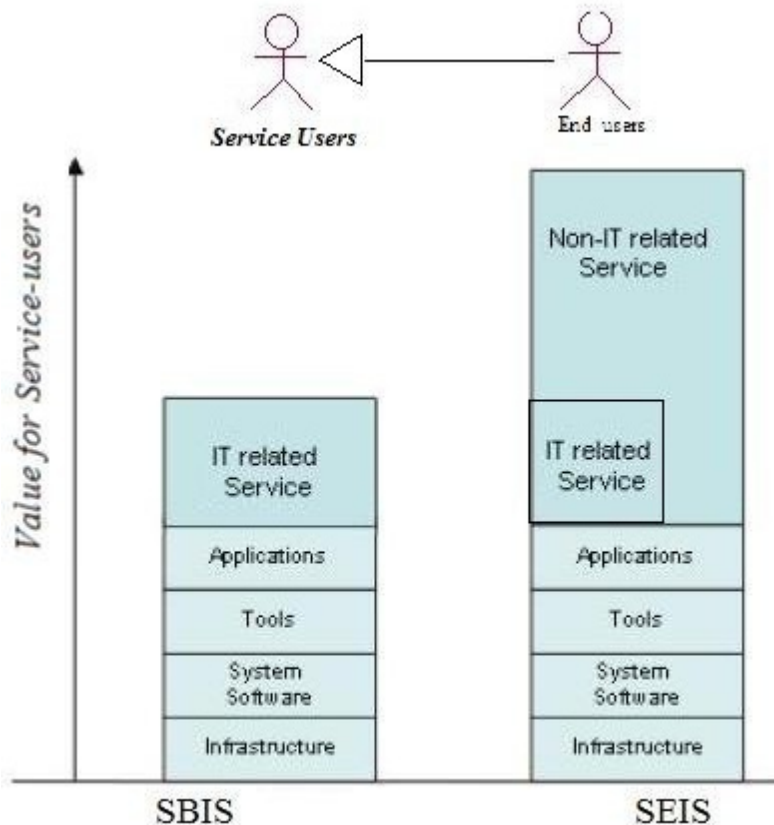


Figure 2.1: Software-based Internet services (SBIS) and Software-enabled Internet Services (SEIS). SEIS may use SBIS.

for the end-users is shown in Figure 2.1. The figure is modified from figure in [90]. The aim of showing this figure is to illustrate that service can be defined differently.

SEIS are services that do not relate directly to IT. The value of such services is mostly associated with the sale of the service itself, not with the software functionalities. The end-users can use the service as it is. In contrast, SBIS services cover capabilities for which the value to the service users is intrinsically related to the IT resources that are delivered via the Internet. The services cannot be used by end-users directly. To be able to use the service, a service integrator must compose the SBIS as SEIS. SBIS includes services in the Cloud computing: Software as a service, platform as a service, and Infrastructure as a service, see the ones that were presented in Section 1.1.

Although SBIS and SEIS rely strongly on the same infrastructure (i.e., hardware, software systems and applications) their value can be different. From the perspective of software developer, SBIS is more valuable since they can be composed

to gain value-added services. The composition can be done to promote (create) new combined functionalities that a single service does not have.

In this thesis, we consider only the SBIS. In the context of the Internet of Services, SBIS are embedded on devices with different levels of device hierarchy, from sensors and actuators, to enterprise levels. The services provide functionalities that can be used by service users. From the software developer perspectives, conceptually, the development of software applications can be done by composing SBIS services. For this, the Service-oriented architecture (SOA) [28] can be adopted for the developing of software applications in the Internet of Services.

Using SOA, architecturally, software applications are built from compound, heterogeneous, autonomous software units called services. If it is the case, service compositions will be a common approach for the development of software applications in the Internet of Services. Software systems and applications are becoming service-based. We call this a service-based application.

There are two types of service-based application. The first type is a service-based application that combines services but is not to be provided as a new service, and the second type are those service-based applications that provide the new combined functionality as a new service. We call the second type composite services.

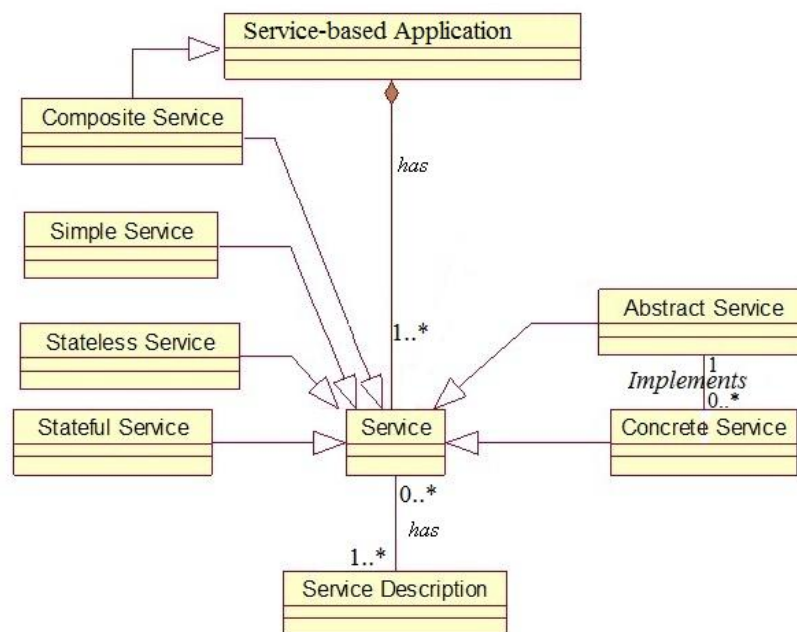


Figure 2.2: A conceptual model of service-based applications. New services can be created by providing the new composite functionalities (composite service) as a new service.

A conceptual model of a service-based application is presented in Figure 2.2. From the figure, it can be seen that a service-based application may use more than one service.

Different approaches and methods may be used to the creation of composite services. Obviously, basic understanding about a service is required. As it can be seen in Figure 2.2, a service may have one or more service description. It describes, for examples, what a service can do, and where the service is located.

It is shown also that a service can be classified either as *Simple* or *Composite*. In this thesis, all services that will be composed are considered as a simple service. As mentioned earlier, a composite service is a type of a service-based application. It composes services and provides the combined functionality as a new service.

A service can also be classified either as *Abstract* or *Concrete*. An abstract service may have one or more concrete services or it may mean that the service will be implemented in the future. But for the composition of services, this thesis considers only *Concrete* services (run-time services).

Furthermore, services can also be classified either as a *State-less* or *State-ful* service. Web services [69] can be considered as an example of stateless services, while UPnP services [46] can be considered as a kind semi state-ful services. UPnP devices use state variables to store the states of specific variables and inform those state changes to other UPnP devices.

2.2 Challenges and Problems

Considering the use of model-driven approaches for the creation of composite services, and the deployment of such composite services on personalized and embedded devices, there are several issues that lead to challenges and problems. In this section, we present five issues. From these five issues, seven research questions are presented in the next section.

2.2.1 Different Roles and Different Service Users

Service development, in principle, involves solution architects, service developers, and service integrator roles. Solution architects and service integrators are the ones who are involved in the service development (i.e., composition at design-time). The end-user, that is a special type of service users, is not included in the development process. But to some extent, this type of service users is the one who have a possibility to compose services at run-time. Figure 2.3 is a use case diagram that shows actors and their roles in the development and use of service-based application.

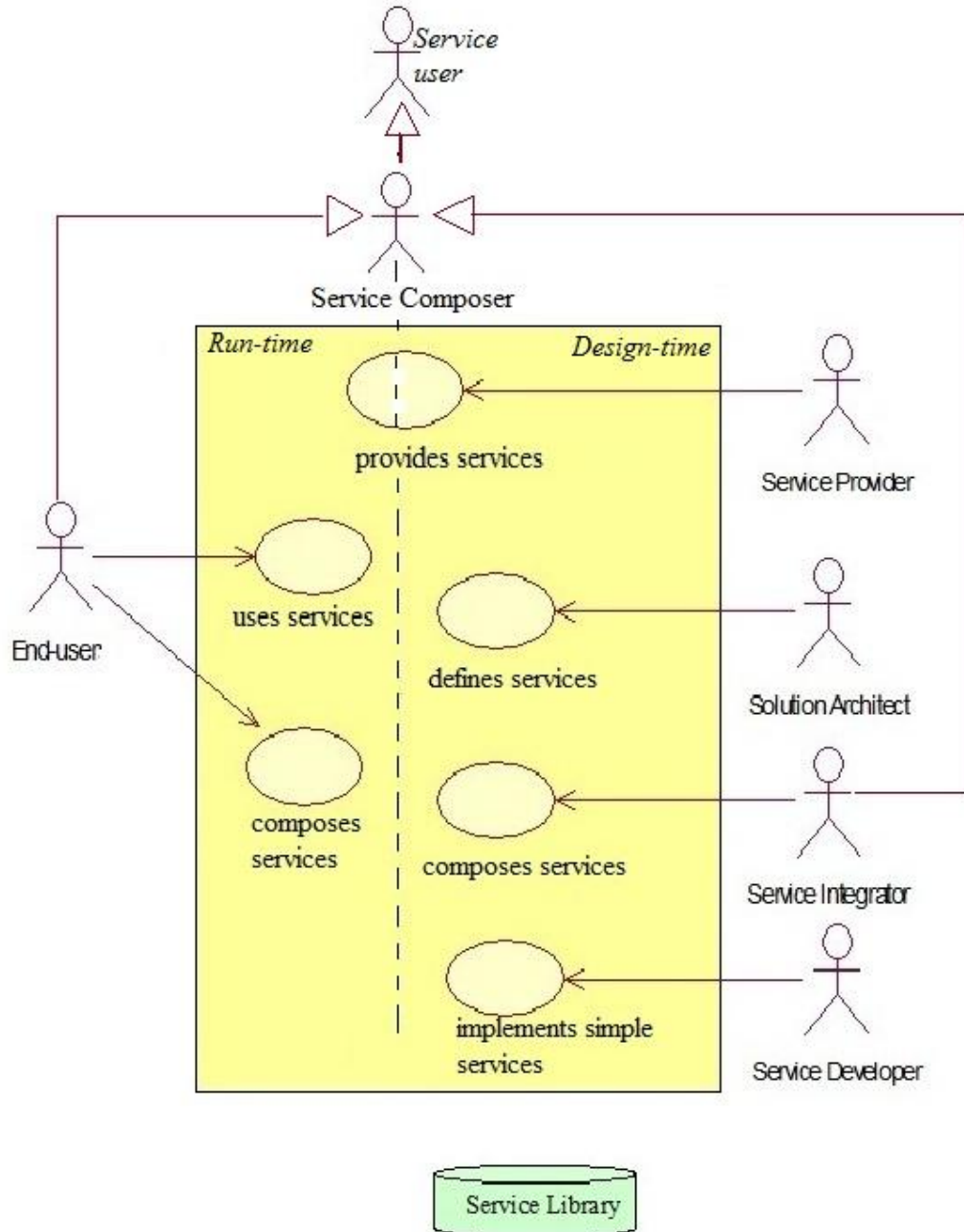


Figure 2.3: Different actors have different roles in the development and use of service-based applications. To some extent, end-users are a type of service composers.

It can be seen in the figure that solution architects use the service library to find existing services that they can use as building blocks for building service-based applications. They may also use service descriptions they find in the service library to populate the set of building blocks. If they require a service that does not exist in the service library, they can publish service metadata that describe interfaces of the services. Service developers will then implement and deploy the required services.

Service developers are the implementers of simple services. They implement (hard coding) new services using programming languages. However, the simple service may include functionalities from existing services as part of their implementation. For this, they can browse the service library to find the required services they could involve in the new services. Moreover, they can use the service library to analyze the impact an envisioned change on a service might have on other services.

Service Integrators assemble solutions from new or existing services. A service integrator is a special type of service composers. The service integrators are the ones who create composite services while the end-users are the ones who use the created services. They compose services at design-time by the use the service library to find building blocks that they can use in their composite applications.

For the service composition, the different roles in Figure 2.3, may require different service presentations (service models). For example, business analysts (i.e., system architects roles) that are less interested in IT-level technical details of services might need to present a service as a building block with high-level information about the service. For the service integrators, complete information about a service may be more important than the service presentation. For end-users, since they are interested in the service composition at run-time, the use of a picture to present a service may help to easily relate a service with the actual device.

Presenting run-time services using graphical representations for different service users is a challenge. With regard to the use of model-driven approach for the development of service-based applications, questions about what service model should be used to present run-time services at design-time and how the presentation processes can be done are necessary to be answered.

2.2.2 Various Implementations of Service-oriented Systems

There exist different standards, implementations, and adaptations of the concept of service-oriented systems. Consequently, they may propose and use different conceptual service models. In this sub section we present a discussion that focuses only on what is a service and how a service is described. For this purpose, we present two examples of conceptual service models copied from [47] and [103]. The first exam-

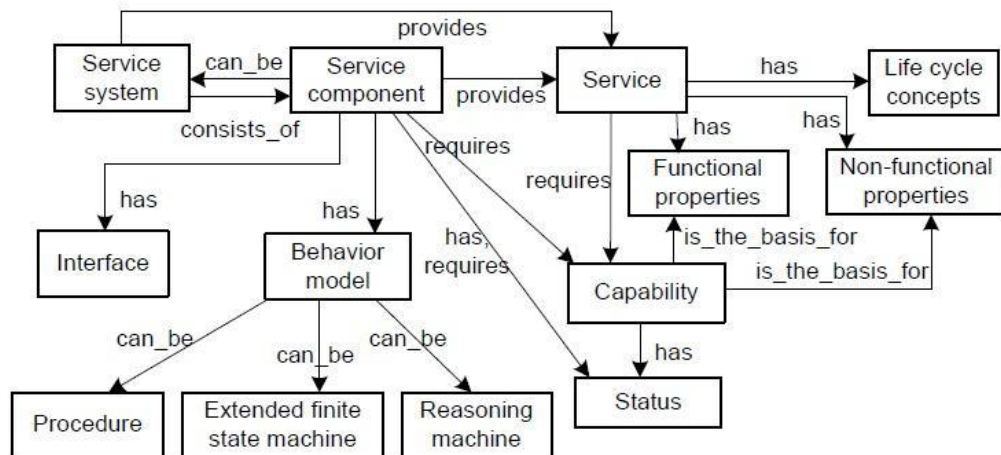


Figure 2.4: A conceptual model of a service presented in [47]. In this model, a service system may be developed by composing service components, since service components provide services.

ple is a conceptual service model presented in [47], see Figure 2.4, while the second example is a conceptual service models presented in [103], see Figure 2.5. The aim is to show that different service-oriented system can be modeled and implemented differently which influences the different ways of describing services. Information about the service, that is a service description, is important for the service users in order to be able to use the service.

A service according to [47], see Figure 2.4, is be provided by a service component or a service system. A service system is a composition of two or more service components. In terms of service-oriented systems a service system is well-known as composite service. The service components itself have behaviors that can be accessed by its defined interfaces.

A service has functional properties, life cycle concepts and non- functional properties. Functional properties describe what a service can do, while the non-functionality properties describe additional information about the services, such as, quality of service, delay, price, availability, time, etc. Functional properties relates to the provided services. Unfortunately, information about these properties is not described in a specific class. In other words, the conceptual service model presented in [47] does not provide any information about the description of a service that is important for the service users in terms the concept of service-orientation systems.

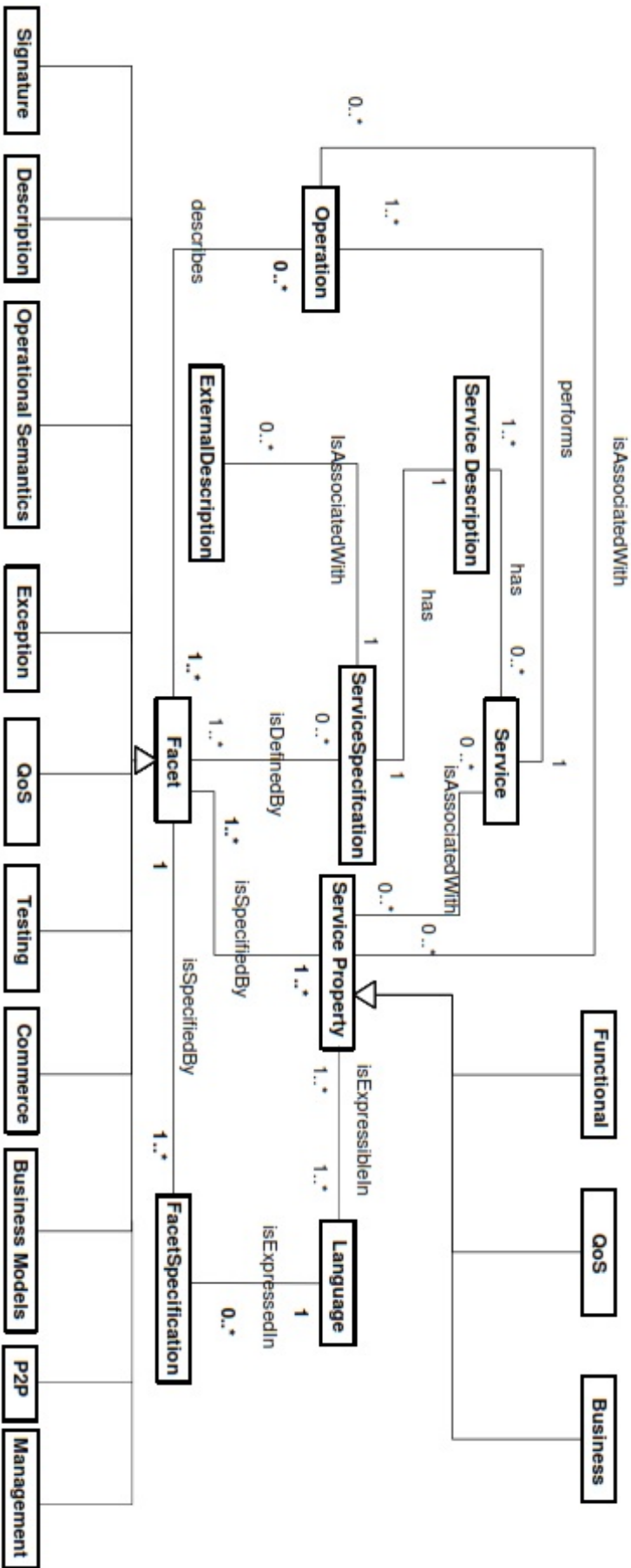


Figure 2.5: A conceptual model of a service presented in [103]. A service has a service description. A service performs operations.

The second example is a conceptual service model that is introduced in [103], see Figure 2.5. A service is associated with service properties which are specified by facets. Moreover, a service performs some operations. Referring to the conceptual service model presented in Figure 2.4, the operations are corresponding to service components that relate to the actual service execution. More explanation about the conceptual service model can be found in [103].

This thesis is interested only on how a service is described in order to service users can use it, and therefore the explanation discusses only this matter. An interesting part of the conceptual service model presented in this figure is that a service may have one or more service descriptions. Generally, a service description specifies its functional attributes such as inputs, outputs, preconditions and effects. As it is also shown in the figure, a service description defines also its non-functional attributes, such as, quality of service, delay, price, etc.

The two examples of a conceptual service (Figure 2.4 and 2.5) show that different conceptual service models exist. In terms of a technology for implementing the SOA concept, different service models also exist, for example SOA Web services and OSGi services. Consequently, different ways, languages and technologies for describing a service exist. For example, there are Web Service description language (WSDL) [69], device profiles for web services (DPWS) [115], XML schema that is used to describe UPnP devices [46]. Originally, the UPnP concept is not related to SOA, but conceptually we can develop service-based applications using UPnP services.

The different implementations of the service-oriented system introduce the interoperability problems. But, it is also a challenge to compose services that were created using different technologies from device levels (e.g., DPWS and UPnP services) to enterprise levels (e.g., Web services). For the success in applying model-driven technology for service creation, a standardized service description is required. The service description should contain information for service categorization. The service categorization will help service users to find similar services or help service integrators to develop scenarios in case when desired services are becoming unavailable.

2.2.3 Various Choices of Modeling Languages

Specifying service interactions is one possible way of building service-based applications. For this, composition techniques and languages are required. Different modeling languages for software and system modeling exist. For example, the Unified modeling language (UML) is a de facto standard for the modeling language. In

the context of Web services, choreography and orchestration languages such as Web Services - Business Process Execution Language (WS-BPEL) [41] are promising steps towards specifying web service interactions and protocols. A BPEL model, representing business processes as an interaction of services, can be executed by using BPEL engines. The BPEL and Business Process Modeling Notations (BPMN) are proposed as a standard language to model business processes.

The existing modeling languages can be used for the current models of software systems. However, the structure of future service-orientation systems will be way more complex than most systems designed and implemented with current service technologies. This complexity is created by several factors such as the need to create flexible and configurable solutions, the need for loose coupling between the services in a service-based application, and the involvement of a multitude of stakeholders and their roles. This particular nature of future service-based applications gravitates towards having to deal simultaneously with dynamicity, openness and global reach. Obviously, this has several important implications to the way of how the services are modeled. Accordingly, a new modeling language is required. The new modeling languages should provide capability to model run-time of a service-based application. This is important since at run-time service, based application should be able to adapt to environment changes. It could be adaptation to, overcoming the mismatches between aggregated services, be able to repair faults, or re-configure the applications in order to better meet their requirements.

However, we will not discuss about such new modeling languages as they are still under development. The point is that, for the modeling of service-based applications it will require modeling languages offering new modeling concepts and suitable notations for modeling software services on different levels of abstraction. The modeling languages should construct abstract away from concrete technologies as much as possible. For this, both existing general and domain specific modeling languages (DSML) can be used. They have their own benefits and drawbacks. DSML supports for fully automated code generation because of its domain specificity feature, but they are only useful on a specific domain. In contrast, general modeling languages may be used for any domain of modeling, but it is often difficult to have a fully automated code generation. There is a question of having a general modeling language but support for an automated code generation. Semantics of the modeling languages are considering the main problem of automated code generation. Incorrect semantics lead to ambiguous machine interpretations. For this reason, research in this area is required. Best practices are sometime the only solution for this issue.

2.2.4 Software Development Method for Embedded Systems

Embedded systems play an important role in the Internet of Things. The embedded systems in this case are all kinds of microprocessor-based device that are able to run a program. In the context of Internet of Services, embedded systems are considered to provide simple services that have been developed by software developers. Mobile phones, microprocessor-based sensor and actuator systems, controllers, microprocessor-based home appliances, and entertainment systems are examples among them. It is not necessarily that the devices have a specific operating system.

Moreover, personalized and embedded devices such as mobile phones are promising devices for end-user interfaces to access services in the Internet of Services. The International Telecommunication Union (ITU) has reported [42] that by at the end of 2010, there are about 5 billion mobile cellular subscribers worldwide. The increasing popularity of mobile devices and embedded networked devices is bringing the promise of pervasive computing [108] closer to reality. Therefore, offering value-added services to mobile users has challenged technology developers, business developers, and mobile services developers.

One of the problems of developing software application for personalized and embedded devices is regarding with the device capabilities and its configuration. On the client side, it is often that when it comes to the deployment, it needs adaptations to which devices the applications will be deployed. For example in the mobile phone domain, the large number of different mobile phones and the significant variability between model of mobile phones makes it hard to create a single application variant that can function correctly across all mobile phones. Not only on the client side, is a problem also found on the server side. On the service provider side, service customizations for each specific device model are also required. In other words, a method of handling device capability and configuration in the development of software embedded systems is required.

2.2.5 Handling Complexity using models

MDD promotes handling of complexity takes using models at higher levels of abstraction and describes software systems using models. However, the main artifact of software systems is an executable program. Therefore, a transformation mechanism of models into executable program is required.

With regard to the use of models in software development there are two perspectives. The first perspective assumes that the models should be a complete model and

is executable. It must describe completely the structure and the behaviors of a software system. Code is only another representation of a model. Having a formal and complete semantic description is the only way to achieve this perspective. This is not a case for the second perceptions. The second perspective assumes that model do not necessary completed since the implementation (into code) is another problem of modeling. With this perspective, source code is not necessary to be generated automatically. It can be manually implemented by software developers or an (semi-) automated code generation to improve the productivity. Code generation by model transformation is the best example of code generation approaches. This is a kind of a vertical model transformation.

High-level modeling constructs abstract away from concrete technologies as much as possible. There is no clue how the mapping of those high-level descriptions into implementation levels and low-level modeling concepts are done. To generate code automatically it is needed an automated mapping that will help to bridge the gap between high-level modeling notations and low-level implementation code which runs on service platforms.

In MDD, code is only considered to be generated from low-level models (i.e., platform-specific models) as the models are often detail and near to the solution. In contrast, in more general context we want to use high-level models to specify software and at the same time we want to have automated code generation. The question is how an executable code can be generated from high-level models in an automated manner.

2.3 Further Research Questions

We have mentioned that the motivation of this thesis is to use the MDD approach for the creation of composite service and their deployment on personalized and embedded devices in the Internet of services. For this, we have analyzed the fundamental question presented in Section 2.1 and identified that the fundamental question covers four areas which are related, as follows:

- A. Model-driven development(MDD)
- B. Creating composite services
- C. Deployment on personalized and embedded devices
- D. The use of existing services.

Out from the four areas above and their relations, we identified five sub problems that have been discussed in Section 2.2, and listed as follows:

1. **A+D: MDD and the use of existing services.** This sub problem has been discussed in 2.2.1.
2. **B+D: Creating composite services and the use of existing services.** This sub problem has been discussed in Section 2.2.2.
3. **A+B: MDD and creating composite services.** This sub problem has been discussed in Section 2.2.3.
4. **C: Deployment on personalized devices.** This sub problem has been discussed in Section 2.2.4.
5. **A+C: MDD and deployment on personalized and embedded devices.** This sub problem has been discussed in Section 2.2.5.

To succeed in applying the MDD approach, all the sub problems listed above must be solved. To guide to the solution of these sub problems, seven sub questions that cover all the identified sub problems are presented. Other sub questions may arise, but it will be covered by these seven questions. Therefore, answer to these seven sub questions implies to answer the fundamental question. More details analysis and the formulated sub questions are presented as follows:

2.3.1 Questions 1 and 2

At the end of Section 2.2.1 it has been discussed the possibility of different types of service users and different roles; service developers, service integrators and end users. In order to service developers and integrators are able to apply the MDD approach for composing services at model levels, run-time services must be presented into service models at design time. For the success in applying MDD, the two first important questions are:

What models should we use to present run-time services for different service users? (Q1)

How can the presentation of run-time services into service models be done? (Q2)

2.3.2 Question 3

Having a complete and formal service description leads to a un-ambiguity interpretation and gives complete information of how the service will be visually presented as models. To be able to present run-time services, the only available information is service description. As it has been discussed at the end of Section 2.2.2, there are

several service description standards and technologies that can be used to describe run-time services. An important question is:

Which service description technology should we use to present run-time services? (Q3)

2.3.3 Question 4

For the modeling of service-based applications, it requires modeling languages. Within the context of MDD, the modeling process should construct abstract away from concrete technologies as much as possible. At same time automation of code generation is one of the indicators for the success of the use MDD for software development. Therefore, the chosen modeling languages influences the success in applying the MDD for software development. It is often only DSL that can provide the automation.

It has been discussed at the end of Section 2.2.3 that different modeling languages are available. A new important question is:

Which modeling languages should be used to model service-based applications? (Q4)

If we can use any existing modeling languages, then it will answer the sub question that somehow implies to answer the fundamental question.

2.3.4 Questions 5

In the software development of embedded systems, information about device capability and configuration may be introduced at the beginning of the development process. In contrast, in the perspective of MDD, this information will be given to the lowest model level, which is often done at the last step of the modeling process. However, as it has been discussed in the end of Section 2.2.4, the variability of targeted devices is also a problem. Since the MDD approach will bring up the problems at model levels, the question is:

To what extent should device capability and configuration be included in the design-time (modeling step)? (Q5)

2.3.5 Questions 6 and 7

The important artifact of software engineering is an executable model which is in this case represented by code. To be able to fully apply MDD for the development of service-based applications, modeling tools must support for an automated code generation. In contrast, as it has been discussed at the end of Section 2.2.5, it is often difficult to have an automated code generation from models at a high abstraction level. If it works, then it is often restricted to specific application areas using domain specific languages (DSL). Two more important questions for the success in applying MDD are:

How can code generation from a high-level model of service-based application be automated? (Q6)

Should we add a platform model or at least some platform patterns? (Q7)

As mentioned earlier, an automation of code generation is one of the indicators for the success of the use of MDD for the development of service-based application. Thus, answering to these questions implies to the answer of the fundamental questions.

Chapter 3

Background

This section presents the background of the thesis. It starts with an overview of the theoretical background of service-oriented systems in Section 3.1. This includes the definition of a service and service descriptions, and an overview of existing service-oriented technologies with its frameworks. Section 3.2 presents state-of-the-art of model-driven development approaches. Brief overviews of embedded systems and software product lines follow in Section 3.3 and Section 3.5. Since the thesis was done in the context of the ISIS project, a brief overview of the project will also be presented in Section 3.5.

3.1 Service-oriented Systems

The service-oriented system is a more general concept than service-oriented architectures (SOA). It is essentially a collection of services that interact with each other to function as a composite system. The interactions can involve either simple data passing or it could involve two or more services coordinating some activities.

SOA represents an architectural model that aims to enhance the agility and cost effectiveness of an enterprise while reducing the burden on IT and on the overall organization. It accomplishes this by positioning services as the primary means by which the solution is represented. The OASIS SOA reference model [73] defines SOA as follows:

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

The definition above indicates that there are two main areas where SOA differs from other approaches to information technologies. SOA reflects the reality that service ownership must be considered in the design of software applications. In the service-oriented systems, services can be provided by third parties. This is something different with regard to the older concepts (such as component-oriented systems) in terms of ownership and control. In contrast with the component-based applications, services will normally be provided by third parties, which are not necessarily the owner of the service-based applications. Moreover, the owner of service-based applications does not have control over the execution of included services.

A number of studies and publications about SOA are concentrating on the definition and implementation of individual (simple) services. However, in the context of the Internet of Services, building enterprise solution(s) typically requires composing multiple existing services at any levels from device services to enterprise services. Moreover, these composite services can again be composed with other services, in a recursive manner. Such recursive service composition is considered as one of the most important SOA features. This feature allows to rapidly build new solutions based on existing pre-made services that can be from third parties. As the amount of simple and composite services grows, the easier it becomes to implement new enterprise solutions.

3.1.1 Definition of a Service

A service can be defined in different ways. In [89] for instance, a service is defined as *asset of functions provided by a (server) software or system to client software or system, usually accessible through an application programming interface*. Similar definitions as the one above appear in the context of middleware technologies such as Jini [19], .NET [48], or JXTA [112]. These definitions recognize services as a central element in the system implementation.

In this thesis, referring to [3] and [62] a service is defined as a model of software unit. To get an overview of this definition, we have to look at the history of managing the complexity of software systems. It started with the use of software component for managing the complexity of software system and their development. A component-oriented architecture is considered as the solution for the complexity problem. Szyperski [101] stated that components are the way software technology has to go because all other engineering disciplines have introduced components as they became mature. Following, [88] stated that software components and appropriate composition mechanisms provide the means for systematic software reuse.

In general, a software component can be defined as a software part that must be composed with other component using software composition systems to form a final system. A software component can be defined also as an element of a component framework. Although this definition seems to be circular it captures an essential characteristic of components. Components of a system or of a modular system are only components because they have been designed to be assembled and composed with other components. Thus, a basic (and single) component that does neither belong to a component system nor is composable in any way is a contradiction in terms. More importantly, a component cannot function outside a defined framework.

Within the software engineering discipline, a software component model is representing a software unit. The idea of using software component as defined in [62], can be considered as the birth of today's software component and can be seen as an architectural approach of building software systems. Figure 3.1 illustrates a historical perspective of the use of different models to represent a software unit.

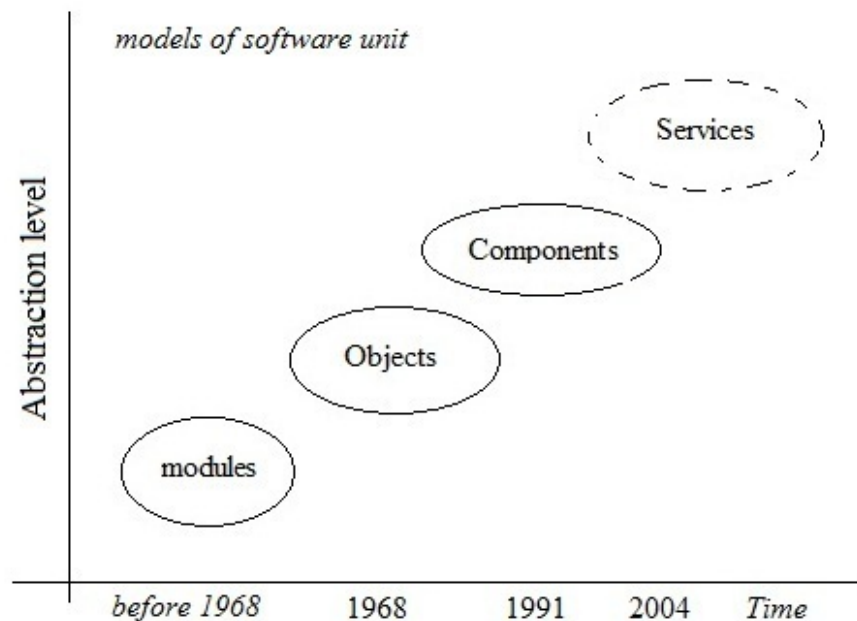


Figure 3.1: The historical perspective of the use of different models of a software unit with regard to the abstraction levels of software unit. A service is seen as a new model of a software unit.

With regard to the software component introduced in [62], a component was viewed as families of routines which are constructed based on so-called rational principles so that these families fit together as building blocks. McIlroy stated that

these families constitute components which are *black-box* entities. Software abstraction, encapsulation and reuse are the key points. Software units can appear in different software component models.

When Assembler was the only available programming language, a routine was considered the first model of software units. As the complexity of software systems was increasing a new model of a software unit called a module was introduced. A module is a simple model. However, a module is more abstract than a routine. Accordingly, objects, components and services can also be seen as models of a software unit. The differences between them are their abstraction levels, means of encapsulation and ownership.

With software component-orientation in mind, a single software component might not work as an application. Therefore a composition system is needed. With this idea, architecturally, software systems that were built by composing pre-made modules are called module-based systems. Accordingly, we have object-based systems, component-based systems and service-oriented systems for the objects, components and services.

According to [3], a software composition system has three aspects: component models, composition techniques and composition languages. Depending on the models of software unit, see Figure 3.1, different composition techniques and languages are required. These two aspects have influenced the development approaches and paradigms. For example, when we use objects as a model of a software unit to build a software system we call the paradigm object-oriented development. Accordingly, we have component-oriented development for component oriented-systems and service-oriented development for service-oriented systems.

3.1.2 Services Visual Representation

Different possible ways can be used to visually represent a service. It can be a graphical or textual visually representation. These different ways may use different languages. The languages can be a domain specific language (DSL) or a general modeling language such as UML.

It must be noted that the visual service representation is only the structure part. The internal behavior is invisible, but they can be accessed through the defined ports. The structure is connected to the real implementation of the service (code). The code itself is only a proxy to the actual service implementation that resides in a service provider. In this section we present an example of visual representation of a service using three different graphical representations (of three different modeling languages).

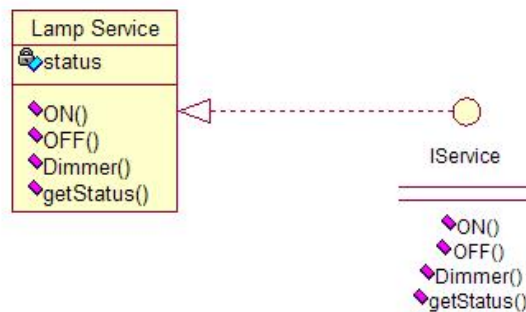


Figure 3.2: An example of a service presentation using UML classes.

- UML Classes.

The first example of graphical representation of a service is using a general modeling language, UML. As a general modeling language, the UML can be used to model any things, including services. For example, in [71] UML is used to model Web services, by presenting Web services as a class diagram.

Figure 3.2 shows an example of graphical representation of a service using a class diagram. The problem of using class diagram is its capability to describe the dynamicity of the services since a class diagram can be used only to represent the structure which is static. For this, interfaces are used to represent that a service can be used by other classes. Internal implementation of the service is invisible for the service users.

- SCA Components.

Service Component Architecture (SCA) [40] is another example of possible ways to represent a service. SCA is a set of specifications which describe a model for building software applications. An SCA component has interfaces and references, where we can use them to model service-based applications. A SCA component consists of a configured implementation, where an implementation is the piece of program code implementing software functionalities. SCA emphasizes the decoupling of service implementation and of service assembly from the details of infrastructure capabilities, and from the details of the access methods used to invoke services. One basic artifact of SCA is a component, which is the unit of construction for SCA. A component consists of a configured instance of an implementation, where an implementation is the piece of program code providing business functions. In the context of enterprise services, the business function is offered for use

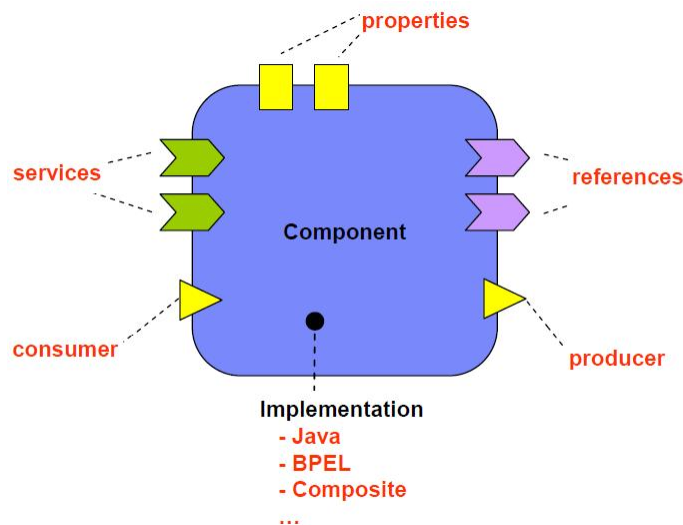


Figure 3.3: An example of service presentation using a SCA component. SCA describes events, operation, and parameters. This promotes both request-reply based and event-based applications modeling. The figure was taken from [40].

by other components as services. Figure 3.3 shows an example of the use of SCA component to represent a service.

SCA allows for a wide variety of implementation technologies, including "traditional" programming languages such as Java, C++, and BPEL, but also scripting languages such as PHP and JavaScript and declarative languages such as XQuery and SQL.

Implementations may depend on services provided by other components. These dependencies are called references. Implementations can have settable properties, which are data values which influence the operation of the business function. The component configures the implementation by providing values for the properties and by wiring the references to services provided by other components.

With SCA components, a service-based application is defined as interactions of SCA components using the wiring techniques. But unfortunately, SCA does not explicitly specify events and therefore it is difficult to model service-based systems that use event-based interactions.

- SoAML Participants.

SoaML [79] is a specification for the UML Profile and Meta-model for Services. SoaML is a standard extension to UML 2 that is meant to facili-

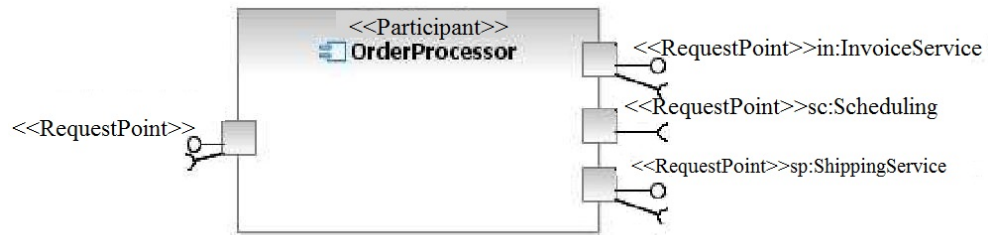


Figure 3.4: An example of service presentation using a SoAML participant. The picture is taken from [79].

tate services modeling. A basic element of SoAML is a participant. A SoAML participant is the type of a provider and/or user of services. As it is considered an extension of UML class, a SoAML participant uses a stereotype `<<Participant>>`. Accordingly, it also uses a stereotype `<<RequestPoint>>` to describe the provided functionality.

In the business domain a SoAML participant may be a person, organization or system. In the systems domain a participant may be a system, application or component. Figure 3.4 shows an example of the use of SoAML participant to represent a service.

Services are provided by participants who are responsible for implementing and using the services. Services implementations may be specified by methods that are owned behaviors of the participants expressed using interactions, activities, state machines, or opaque behaviors. Participants may also delegate service implementations to parts in their internal structure which represent an assembly of other service participants connected together to provide a complete solution, perhaps specified by, and realizing a services architecture.

However, events which are fundamental properties of service-based are not described in the SoAML specification. This means that the SoAML cannot be used to model service-based applications that use event-based style of interactions.

3.1.3 SOA Implementations and Similar Technologies

The openness of the concept of service-orientation systems leads to different ways and technologies for the implementation and adoption of the concept. The different implementations of service-oriented systems may use different ways and technolo-

gies of describing services. They may also use specific description languages. A universal service description language (USDL) is often used to describe services. In this section we present examples of service-oriented technologies that include its related frameworks and API.

A common framework and APIs identify specific functions that need to be addressed in order to achieve decentralized interoperability. The framework does not determine the particular technologies used to fulfill the functions but rather divides the problem space into sub-problems with specified relationships. This functional decomposition allows differing solutions to sub-problems without overlaps, conflicts or omitted functionality. This is not to say that all applications must offer the same facilities, rather that when a feature is offered it should fit into a common framework and preferably have a standard expression.

3.1.3.1 Universal Plug and Play, UPnP

The Universal Plug and Play (UPnP) [46][32], technologies are not neither an implementation nor adoption of service-oriented architecture concepts. However, their concepts are similar in which a service is a basic entity. In terms of SOA, UPnP control points can be seen as service users while UPnP devices can be seen as service providers. There is no service registry in the UPnP technology. However, a specific UPnP control point can be built to function as a service registry. A model of UPnP devices can be described in a UML class diagram as shown in Figure 3.5.

The UPnP specifications can be divided into: Architecture and Profiles. The UPnP Device Architecture (UDA) is the core upon which the device and service specifications are built. The UDA defines two types of hosts: control points (which are service users) and devices (service providers). UPnP devices may have several services. UPnP devices may also be embedded within other devices. In addition, UPnP technology has very strong roots in standard Web technologies. As explained previously in this chapter, the UPnP is based on IP, HTTP, XML and SOAP.

Using UPnP services, new composite applications can be built by composing available UPnP services. Moreover, the new application can be provided as a new UPnP services in an incremental manner.

There several UPnP framework and API available in the market. This is because UPnP specification is open and uses open network protocols. By providing a set of defined network protocols, UPnP allows devices to build their own APIs that implement these protocols - in whatever language or platform they choose. One example of UPnP frameworks is Cybergarage from Cyberlink [52]. The framework provides an environment for building UPnP devices and its services. Other example

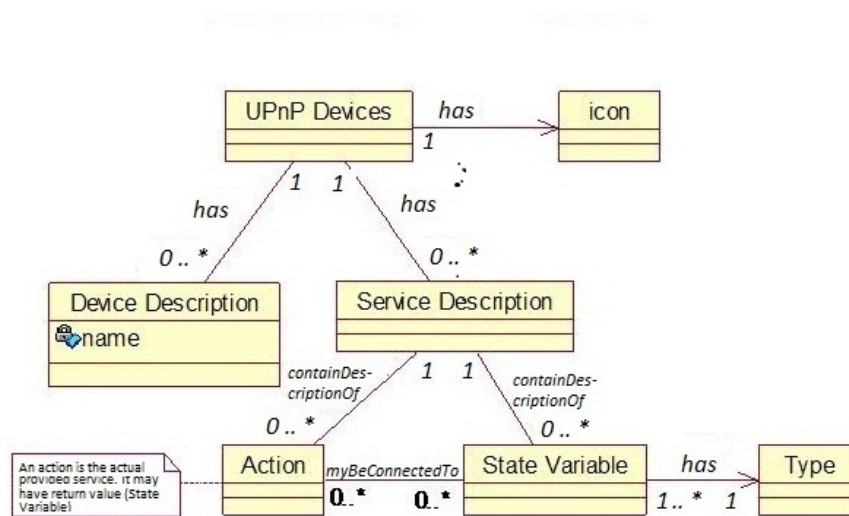


Figure 3.5: A conceptual model of UPnP devices presented in a UML class diagram. A UPnP device has two different descriptions: device and service descriptions.

is a framework from Intel that based on C++ and run on both Linux and Windows.

3.1.3.2 Web Services

The popularity of Web Services [69] has fostered the massive adoption of the SOA paradigm. However, despite its huge success, it is important to clarify that Web Services are only one type of possible implementation of SOA concept. A Web service is described using Web service description language (WSDL). Thus, A WSDL document defines services as collections of network endpoints, or ports. Web services can be modeled using a class diagram shown in Figure 3.6.

A **Service** is a collection of related endpoints. The abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types** which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A **binding** is a concrete protocol and data format specification for a particular port type. A port class is defined by associating a network address with a reusable binding, and a collection of ports define a service.

WSDL also describes **Operation**, an abstract description of an action supported by the service. This is the actual service functionality that can be accessed through a

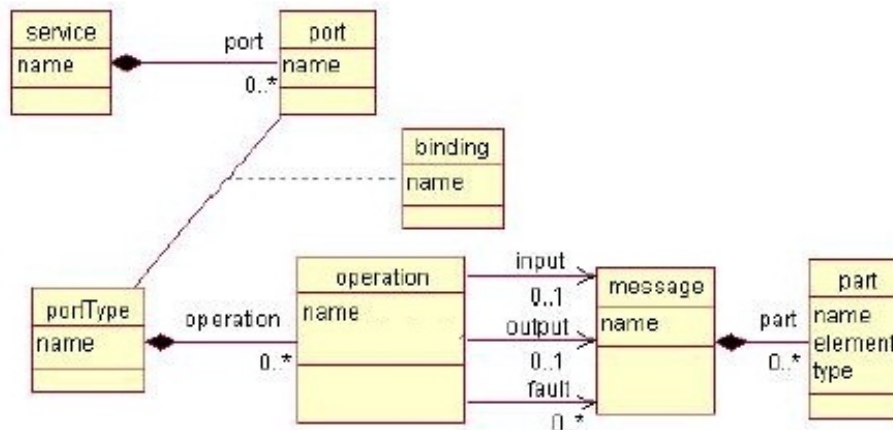


Figure 3.6: A meta-model of Web service description presented in a UML class diagram.

Port, a single endpoint defined as a combination of a binding and a network address.

Similar with UPnP, the Web services stack is composed of several protocols, such as HTTP, XML, and SOAP. In addition, the stack makes use Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI). The WSDL is used to describe the technical details of how a Web service can be accessed and invoked over the Web. The description details technical requirements such as Internet addresses, ports, service names, argument names, and data types used by a Web service. Thus, it emphasizes on the technical and implementation aspects of a service.

A WSDL describes the service that can be used by a service requester. However, in contrast with UPnP services that embed on devices, Web services were designed for enterprise services. Within the context of the Internet of services, this leads to the heterogeneous services from device levels to enterprises service levels. Figure 3.7 shows an example of (a part of) a service description in WSDL.

For the framework, there are several available frameworks such as Apache Axis, Java for Web Service, Web Service Invocation framework, JSON-RPC-Java, XFire. Among them are open sources. The framework provides a simple, easy and fast development of Web services.

In Web services technology, in order to be able to invoke a Web service, a service user (client) must implement a code for service invocations. A useful class in Apache Axis is `wsdl2Java` class that generates Java classes from an existing WSDL document. The generated classes represent client stubs, server skeletons and data

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions name="power_service"
.....
  <message name="serviceRequest">
    <part name="status" type="xsd:string"/>
  </message>
  <message name="serviceResponse">
    <part name="status" type="xsd:boolean"/>
  </message>

  <portType name="device_PortType">
    <operation name="setPower">
      <input message="tns:serviceRequest"/>
      <output message="tns:serviceResponse"/>
    </operation>
  </portType>

  <binding name="device_Binding" type="tns:device_PortType">
    .....
  </binding>

  <service name="device_Service">
    <port binding="tns:device_Binding" name="device_Port">
      <soap:address
        location="http://localhost:8080/soap/servlet/device"/>
    </port>
  </service>
</definitions>

```

Service

Figure 3.7: An example of Web service description. The names of the services (operations) are listed in the description.

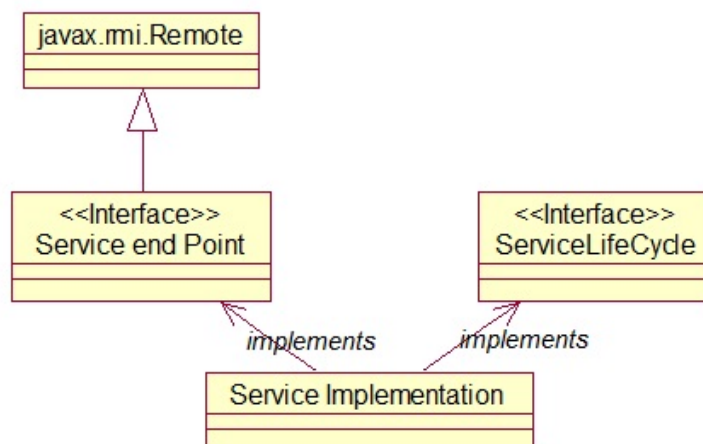


Figure 3.8: An example generated classes from WSDL2Java in the AXIS web service framework. The classes are used to invoke the actual services that reside on a service provider.

types that will help service developers to implement code for service invocations on the client side. Figure 3.8 shows a Web service end-point interface generated by the `wsdl2Java`. As it can be seen in the figure there is a service endpoint implementation (concrete) class and a service endpoint interface definition.

3.1.3.3 Devices Profile for Web Services, DPWS

Devices Profile for Web Services (DPWS) [115] [74] is a Web Services profile that enables plug-and-play for networked devices. The purpose of DPWS is similar to UPnP, but it employs the popularity of Web Services. Similar with UPnP, a service user (PC or other devices) can detect DPWS-enabled devices on a network, then discover and invoke the provided service functionality.

DPWS is based on existing Web Services standards, including XML, WSDL, SOAP, MTOM, and HTTP. Current version (2010) of DPWS was based on the core Web Services standards that include WSDL 1.1, SOAP 1.2, XML Schema, WS-Addressing, and further comprises WS-Policy, WS-Security, WS-Discovery, WS-Eventing, WS-Metadata Exchange and WS-Transfer. With these standards, DPWS supports dynamic discovery mechanism. Just like UPnP, DPWS also supports for home automation environments such as an application to communicate security cameras, a motorized television mount, etc.

Within the context of Microsoft technology, DPWS for the .NET Micro Framework also incorporates this functionality. For example, there are classes in the .NET

Micro Framework that are used for reading and writing XML documents, including the XML parsing functionality. With these classes the service implementations and the development of DPWS-based applications can be easier.

DPWS concept provides the following functionality between compatible devices:

1. Discovering DPWS-capable devices and the services they offer
2. Sending messages to DPWS-capable devices and receiving replies
3. Describing a Web service by providing a WSDL file
4. Interacting with a service using its description
5. Subscribing to and receiving events from a Web service

Within the context of service-oriented system, devices can be DPWS clients (service users), servers (service provider), or both. DPWS specifies an architecture in which devices run two kinds of services that are hosting services and hosted services. The hosting services are a service that are associated to a device, and is a key important part in the device discovery mechanism. The hosted services are mostly functional and depend on their hosting device for discovery.

DPWS also specifies a set of built-in services for service discovery, eventing, publishing, and meta-data. The discovery service is used by a device to advertise itself and to discover other devices. The Metadata exchange services provide dynamic access to a device's hosted services and to their metadata. The publish/subscribe eventing services are allowing other devices to subscribe to asynchronous event messages produced by a given service.

An example of DPWS framework is the Microsoft .NET Micro Framework [50]. The framework is a powerful and flexible platform for rapidly creating embedded device firmware with Microsoft Visual Studio. The framework brings essential benefits of the Microsoft .NET platform to devices that do not need the full functionality that is available in the .NET Framework and the .NET Compact Framework.

Another example of DPWS framework is the Web Services for Devices (WS4D) framework [115]. The WS4D framework is Java-based framework. Originally, it applies the Service-Oriented Architecture (SOA) and Web services technologies to the application domains of industrial automation, home entertainment, automotive systems and telecommunication systems. WS4D advances results of the ITEA project SIRENA and is managed by the University of Rostock, University of Dortmund and MATERNA [115].

WS4D is all about using internet technologies like XML, HTTP and Web Services to connect resource-constrained devices in ad hoc networks and still conserve

interoperability with Web services as specified by the W3C. This enables the usage of high level concepts for Web services also in low level distributed embedded systems. So WS4D provides technologies for easy setup and management of network-connected devices in distributed embedded systems.

3.1.3.4 Open Service Gateway initiative, OSGi

The OSGi Alliance was originally initiated and founded by Ericsson, IBM, Motorola, and Sun Microsystems in March 1999 [60]. Among other members of the alliance are (as of May 2007) big companies from quite different business areas, for example IONA Technologies, Deutsche Telekom, etc.

The OSGi service platform brings service-oriented architecture principles to single nodes. OSGi has evolved from its initial concept as a residential gateway to becoming the Java-based. A residential gateway is often used in smart homes to control/manage different devices (and their services). Using OSGi framework, devices (and their services) can be easily installed, run, stopped, and installed.

Deployable OSGi components are called bundles. A bundle is composed of a jar archive with additional metadata for clearly defining the requirements for the component to work and the additional resources offered to the platform. The OSGi platform defines a local service registry where bundles can register services and query for implementations. Dynamic bundle interactions are performed through services. This way, the SOA paradigm can be adopted also inside the applications.

The description of the service in OSGi is specified by a Java interface and a set of properties. All services provided by a bundle are declared in an Export-Package statement, as shown in Figure 3.9 below.

```

Bundle-Version      1.0.0
Bundle-ClassPath    .
Bundle-Activator    DeviceService.Activator
Ant-Version          Apache Ant 1.7.0
Created-By           Selo
Bundle-Name          OSGi Device
Manifest-Version     1.0
Bundle-Description   Simple bundle power service.
Export-Package       org.isis.uia.device.service.power

```

Figure 3.9: An example of OSGi service description presented in XML. The actual service description can only be seen at run-time.

OSGi can be considered a mini application server comprising a rich service-orientation system. OSGi is the global standard for creating and assembling modu-

lar software built with Java technology. It provides a modular services framework and component based architecture. OSGi has its roots in embedded home gateways but matured to a technology that is being adopted in various markets such as automotive, telematics, mobile, e-Health/assisted living and last but not least, Java enterprise servers. One of the smallest OSGi solutions is ProSyst mBS that uses embedded low power hardware with an ARM9 CPU at 156 MHz and 8 MB RAM and flash memory.

Software components (i.e. middleware, apps, services, etc.) can be plugged into the OSGi framework at any point in time and each component has its own lifecycle. In other words, OSGi enables concurrent multi-services and multi-application execution in just one virtual machine. Moreover, all components and APIs are fully manageable and accessible from remote.

The core component of the OSGi Specifications is the OSGi Framework. The Framework provides a standardized environment to applications (called bundles). The Framework is divided in a number of layers. Different vendors have implemented the framework. Three well-known OSGi frameworks are Equinox (www.eclipse.org/equinox), Felix (www.felix.apache.org) and Knopflerfish (www.knopflerfish.org).

3.1.3.5 Jini

A Jini system [19] is a distributed system based on the idea of federating groups of users and the resources required by those users. The focus of the Jini concept is to make the network a more dynamic entity that better reflects the dynamic nature of service-oriented architecture. This is done by enabling the ability to add and remove services flexibly.

Jini uses the term federation to imply coordination between devices with embedded services. In this context, a federation is defined as a collection of autonomous devices which can become aware of one another and cooperate if necessary and valuable. To achieve this, a Jini subsystem implements a set of lookup services that maintain the dynamicity of information about available devices.

A service description in Jini is similar to OSGi in which the service is defined by an interface. Jini is strong related to the SOA concept where the dynamicity of services is important behaviors. Jini has the same concept of a service registry, a service provider, and service users. Jini devices can act as service provider and service users.

Jini's architecture is based on the principle of that the system is dynamic. This is contrast to other SOA technologies such as J2EE and .NET. They use a static

approach to installing the software with its appropriate stubs and skeletons. One of Jini's strongest features is downloadable service-proxies that communicate with the Jini service.

Jini from Sun Microsystems, underneath all the hype, is but a coordination framework evolved and adapted from academic research and tailored specifically to Java. The original inspiration comes from the works of David Gelernter (Yale University) and Nick Carriero who built the Linda [14]. An inspired result was the JavaSpaces technology with its later evolution into Jini.

PRINCEPS [21] is a Jini adoption that makes it possible to dramatically improve service flexibility by federating services in a dynamic and self-healing networked community. The framework provides the clients with a (web-based) mechanism for selecting services according to functional requirements (i.e., the service interface) and non-functional requirements (the quality of service, i.e., reliability, performance). PRINCEPS provides service substitutability at various levels. In fact, any kind of service can be plugged in PRINCEPS: CORBA-based services, socket-based services, and centralized services. PRINCEPS also supports service developers with its own mechanism to integrate existing services. Preliminary performance experiments and lessons learned from the PRINCEPS prototype are also discussed.

3.2 Model-driven Development

There exist different approaches, techniques, and languages for the development of software systems. Model driven development (MDD) is a kind of innovative software development that meets general industry requirements for software development, such as reduction the operating costs, reductions in time to market, and the need for open solutions. One example is OMG Model Driven Architecture, MDA [64]. Figure 3.10 shows the model-driven development approach proposed by OMG.

Software development using OMG's MDA framework implies creation of models of the following kinds (see Figure 3.10): the computation independent model (CIM) at the business system model level, the platform independent model (PIM) at the information system model level, the platform-specific model (PSM) at the software model level, and finally, the code which will automatically be generated from the PSM at the software model level.

3.2.1 Models and the Use of Models

In general, a model is an abstract representation of a system and it might come in different abstraction levels. The higher the abstraction level the less the information the model contains, since abstractions will only present the essential information. Different *view points* or *model views* might be used to model a system, as shown in the following equation.

$$\text{System} \equiv \bigcup_{i=1}^n \text{modelview}_i$$

$$\text{modelview} \Rightarrow (\text{abstraction}_i, \text{notation}_i)$$

Different notations can be used to present the different model views at different abstraction levels. Using UML, for example, there are 9 different views of a system that are the structure models (class diagrams, collaboration diagrams, component diagrams and deployment diagram) and behavioral models (activity diagrams, sequence diagrams, use case diagrams, and state chart diagram). For different model views they use different notations.

In software development, models are used to specify software systems. In this

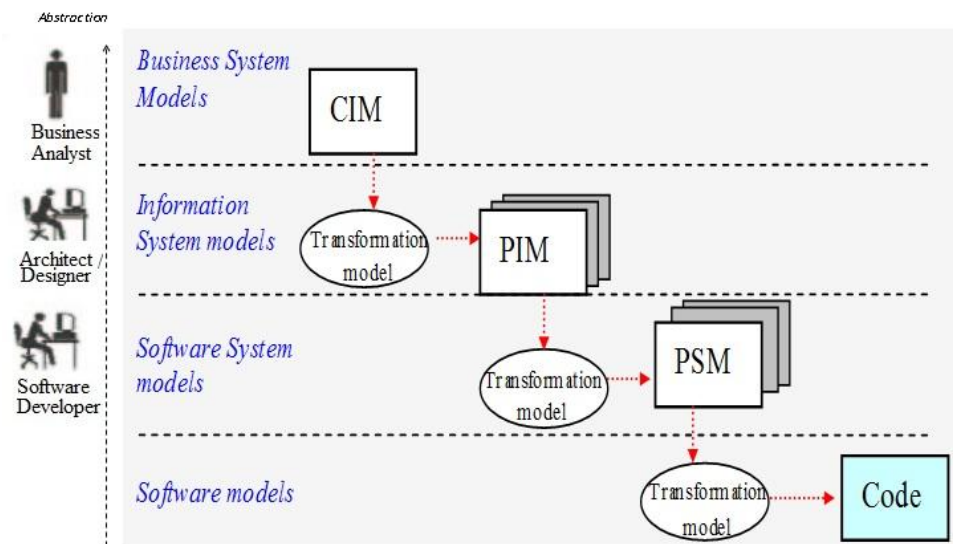


Figure 3.10: OMG's Model-driven architectures (MDA). Different actors have different roles in the development of software systems. The idea is to separate the problem with the solution.

case, models are used to describe the structures and the behaviors of the software systems. The structure specifies what the instances of the model are; it identifies the meaningful components of the model construct and relates them to each other. The structure consists of the classes that can appear in the system as well as their mutual dependencies given as associations between the classes. The structure given by a model is used at runtime to allow the creation of several runtime objects as instances of the structure model. The structure model in UML emphasizes the static structure of the system using objects, attributes, operations, and relationships.

The behavioral model emphasizes the dynamic behavior of a system, which can essentially be expressed in three different types of behavioral models; interaction, activity diagrams, and state machine diagrams. Interaction diagrams are often used to specify how the instances of model elements are to interact with each other (roles) and to identify the interfaces of the classifiers.

The UML activity diagrams are used mostly to model data/object flow systems. It is good to model the behavior of systems which do not greatly depend on external events, mostly having steps that run to completion, rather than being interrupted by events and requiring object/data flow between steps. Activity diagrams in UML 2.0 specified notations communicate with external systems using receive and send signals. For model-driven engineering state-machines diagrams are considered as the most executable models [35]. In [70] [66], it is proofed that for example, sequence diagrams [70] and activity diagrams [66] are transformable into state chart or state machine diagrams.

Formalism is one of the serious problems in the use of models to specify software systems. Message sequence charts (MSCs) [44] constitutes an attractive visual formalism that is widely used to capture system requirements during the early design stages. MSC is a graphical and textual language for the description and specification of the interactions between objects or system components.

A version of MSCs called sequence diagrams is one of the behavioral diagram types adopted in the UML. It can be used to capture the typical inter-working of processes or objects. The standard defines the allowed syntactic constructs rigorously, and is also accompanied by a formal semantics [43] that provides unambiguous meaning to basic MSCs in a process algebraic style. Other efforts at defining a rigorous syntax and semantics for MSCs have been made [13, 58], and some tools supporting their analysis are available [2, 6].

However, it was found that MSCs have two important weaknesses with regard to their using for specifying software system. In [24] it is listed two of the weaknesses. The first is that it proposes a weak partial ordering semantics that makes it

impossible to capture some behavioral requirements. The second weakness is that the relationship between the MSC requirements and the executable specification is not clear. To solve these, [24] also proposed an extension on the MSC standard called Live Sequence Charts(LSCs).

LSCs can be viewed as a multi-modal version of MSCs, with various means for distinguishing between possible, necessary and forbidden behavior. Using LSCs, one can start to look more seriously at the relationships and possible automated transitions between requirements, as captured by use cases and LSCs, and executable specifications, as captured by, for example, state charts.

3.2.2 Modeling Languages

Expressing a software system by a model requires a language. A common approach is by using programming languages. With programming languages, a software system is presented in form of source code. Another way of representing models of a software system is using modeling languages. However, in order to have working software, the models must be transformed into source code. So, in respect to the fact that both programming languages and modeling languages aim to produce code at the end of modeling, then we consider that they are similar. In [99], the similarities and differences between them are discussed. They have identified that modeling languages use higher abstraction, mostly use graphical representation and they are often not executable. In contrast, programming languages are executable, mostly using text representation, and of lower abstraction level.

Different modeling languages can be used to model a system at different abstraction levels and using different representations. It depends on the purpose of the language. It can be a general modeling language or domain-specific modeling language (DSML). UML is an example of a general modeling language. UML addresses the modeling of architecture, objects, interactions between objects, data modeling aspects of the application life cycle, as well as the design aspects of component-based development including construction and assembly. UML is powerful enough to be used to represent artifacts of legacy systems captured in terms of Classes, Interfaces, Use Cases, Activity, Graphs, etc. UML models can be easily exported to other tools in the life cycle chain using XML.

DSL is an executable specification language that offers, through using of appropriate notations and abstractions, expressive power restricted to a particular problem domain. The domain specificity of a language is a matter of degree. Any language has a certain scope of applicability, but some of them are more focused than others.

3.2.3 The relation between Systems, Models, and Languages

In model-driven development, models are used to specify a software system. In this case, models are used to describe the structure and the behavior of the software system. The specification (models) might be an executable or non-executable. In case the models are not executable, code for implementing the software system might be generated either partly or fully automated using code generator.

To describe a software system by models, we need a language. Figure 3.11 shows relations between systems, models, and languages. To be useful, i.e. capable to be used for describing structures and behaviors of a software system, a modeling language should facilitate a way to present the structures and the behaviors (with their constraints) of the software system.

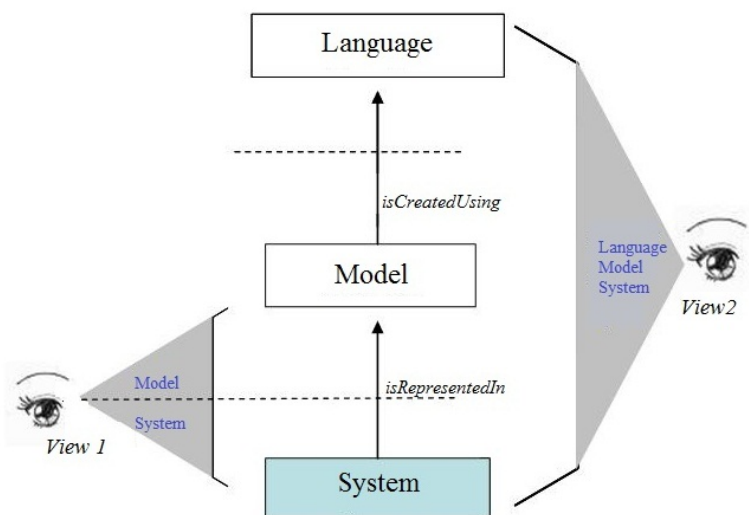


Figure 3.11: An illustration picture showing the relation between systems, models and languages.

It is necessary to note that the term model is relative to the system which is described. To explain the case, we use Figure 3.11 that shows relations between systems, models, and languages. It is depicted in the figure that we might use *view1* or *view2* to see the relations between them. The *view1* is a two levels view (system-model), while the *view2* is a three levels view (system-model-language). Since the term of a system is relative, obviously, we can use the *view1* to see the model-to-language relation as a system-to-model relation, by considering the models in the model-to-language relation as a system while the language as a model.

The fact that we can consider models as a system and describe them with another model, has introduced a concept of meta-model. So, a meta-model is only

another model that describes models and that can be written using another modeling language. In the area of domain specific languages (DSLs), meta-models are often used as an approach for defining a DSL. Interesting question is when a model can be considered as a language. To answer this question, it is meaningful to check how languages aspects are handled in models and meta-models.

3.2.4 Model Transformation

An important aspect of Model Driven Development (MDD) is model transformations, which allows automatic transformations of models into another models. An important emerging standard for transformations language is OMG's QVT (Queries, Views, and Transformations) [78] that is instances of the MOF meta-model. This standard aims to provide a language for expressing transformations between models. Because of the generic nature of MOF, it is also being used as the means of expressing the QVT language itself.

There are four different model transformations; M2M (Model-to-Model transformation), T2M (Text-to-Model) transformation, M2T (Model-to-Text) transformation and T2T (Text-to-Text) transformation. For each of the transformation there it may need a transformation language. Examples for such language are MOFScript

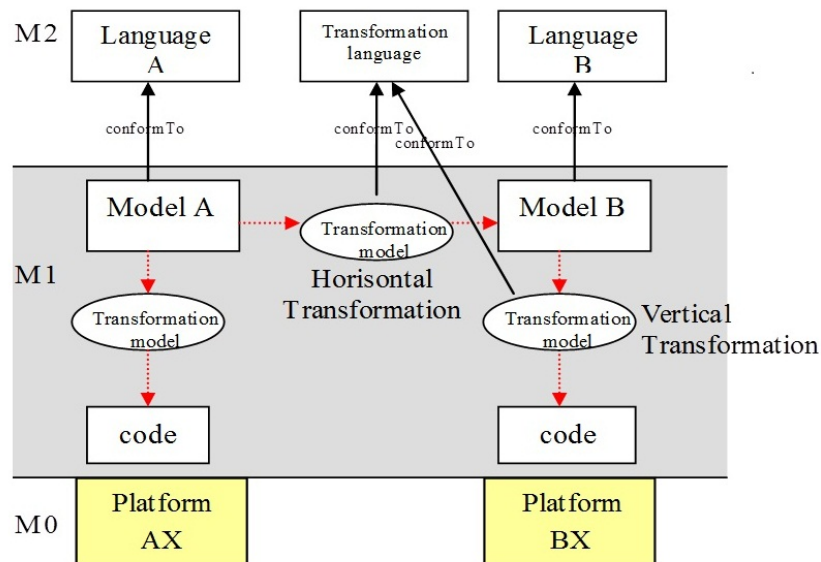


Figure 3.12: A conceptual model of model transformation.

[75], JET and Acceleo for M2T, Xtext and TCS for T2M [27] and ATL [8] for M2M. The T2T transformation is often used for the processing of the M2M, M2T and T2M.

Within the context of MDA, the transformation language is basically used to create model transformations between languages or for model refinements, as model transformation have two main types of transformation; vertical transformation and horizontal transformation. In a vertical transformation the source model is at a different level of abstraction than the target model. Examples of a vertical transformation are refinement (specialization), PIM-to-PSM transformations and abstraction (generalization). Generalization could mean also an abstraction of platforms or a transformation from code into models (reverse engineering). A concrete example of model refinement is a transformation of service models into executable state machines, which is also a model, is presented in [54].

In a horizontal transformation the source model has the same level of abstraction as the target model. Examples of a horizontal transformation are refactoring and merging. In this type of transformation one or more source models are transformed into one or more target models, based on the languages (meta-model) of each of these models. In other words, instances of one meta-model are transformed into instances of another meta-model.

Figure 3.12 illustrates the concept of model transformation. In the figure, model A which is made by a language for platform AX must be transformed into model B in order to code for platform BX can be generated.

3.2.5 Code Generation

An executable program is the most important artifact of software system. Therefore it is important to execute the models that specify the software systems. There are two ways of model execution; model interpretation and code generation.

In order to support the model execution, OMG has developed several additional specifications to the UML that will help tailoring UML to support MDA. Three of these specifications are: *1)* Action Semantics for UML specification that will enhance the language's representation of behavior, *2)* the human-readable UML Textual Notation that will enable a new class of UML editor programs and enhance the way UML models can easy be manipulated, and *3)* standard Software Process Engineering Meta-model that used to define a framework for describing methodologies in a standard way. This standard will not standardize any particular methodology, but will enhance interoperability from one methodology to another.

The realization of model-driven software development requires effective tech-

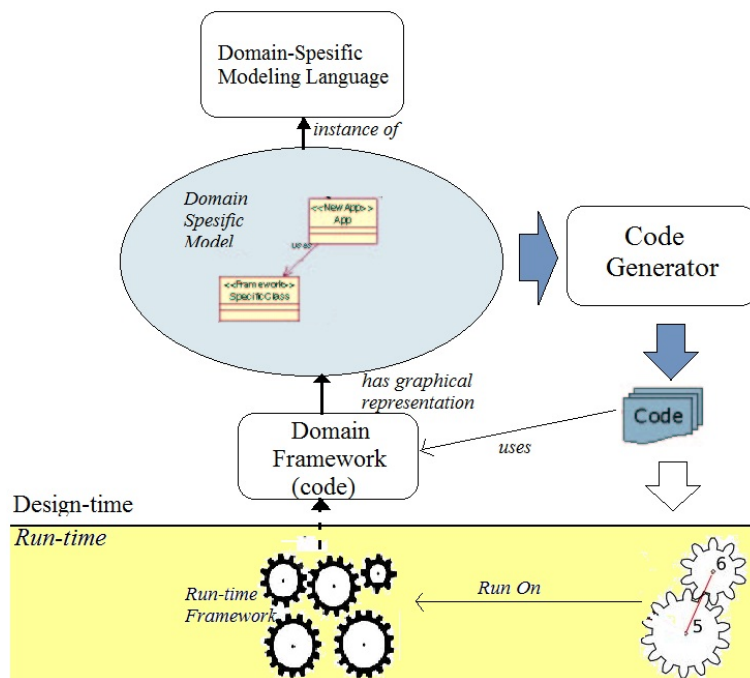


Figure 3.13: A concept of code generation process in the domain-specific modeling languages (DSM). There is a strong relation between the modeling language, the domain framework and code generation.

techniques for implementing code generators for domain-specific languages. In [39], a technique for improving separation of concerns in the implementation of generators is proposed. The core technique is code generation by model transformation, that is, the generation of a structured representation (model) of the target program instead of plain text. Their approach enables the transformation of code after generation, which in turn enables the extension of the target language with features that allow better modularity in code generation rules. The technique can also be applied to internal code generation for the translation of high-level extensions of a DSL to lower-level constructs within the same DSL using model-to-model transformations.

Domain-specific modeling (DSM) provides a modern solution to demands for higher productivity by constricting the gap between problem and solution modeling. In DSM the models are constructed using concepts that represent things in the application domain, not concepts of a given programming language. The most important part of the using of model-driven development method is having running application. In a number of cases the final products can be automatically generated from these high-level specifications with domain-specific code generators. The code generation process in DSM is shown in Figure 3.13 Code generation process in DSM.

The automation in DSM is possible because of domain-specificity: both the

modeling language and code generators fit to the requirements of a narrowly defined domain. Domain-specific languages are a key component of code generation. While we have ample experience building code generators and compilers, modern software developers expect integrated development environments such as Eclipse and Visual Studio to boost their productivity.

There is a 1:1 mapping from every model notation in the modeling language to one unit of code in the framework. Therefore it is possible to achieve a fully automated code generation. An example of code generation method that aligns UML state machines and Temporal logic to generate code from service models is introduced in [55].

In model-driven development, the generated code might have different types for different purposes. It might be in a form of programs (source code), configuration files, scripts, html files, etc.

3.3 Embedded Systems

With the advance of microprocessor technologies, most of today's electronic equipment are microprocessor-based. An embedded system is defined as a specialized and microprocessor-based system that is part of a larger system or machine. For the end-users, an embedded system is a computer system that is integrated within other equipment in such a way that the user is often unaware that a computer system is inside that equipment. Examples of such embedded systems are electronic systems that are installed in cars, mobile phones, home entertainment devices, home appliances, handheld game consoles, and sensors modules.

Some embedded systems might include an operating system, but in many cases they are designed to work in a specialized area such that the entire logic can be implemented as one single program. An example of embedded systems with an operating system is SunSPOT [45]. SunSPOT has a small Java ME virtual machine called "Squawk VM" to run Java classes. With regards to the Internet of Things, embedded devices such as SunSPOT, can help to speed the onset of the Internet of Things by providing networked devices and tools to software developers and researchers.

An embedded software system often provides software functionalities that can be used by end-users. In the context of the Internet of Services and SBSS that have been presented in Section 2.1, the functionalities are not only intended for end-users but it will be also provided as software-based internet services (SBIS).

It is expected that devices in the Internet of Things are able to communicate with

each other. It is therefore important to consider the device communication issues in the design of embedded software system. We are interested in two important issues: device coordination frameworks and communication protocols. We present a short discussion of these issues in the following subsections.

Considering that devices in the Internet of Services provide and/or use embedded services, a coordination mechanism between devices has become an important research area. A number of architectures addressing mobile and specialized embedded services have emerged recently. These architectures are essentially coordination frameworks that propose certain ways and means of device interaction with the ultimate aim of simple, seamless, and scalable device inter-operability. Device coordination essentially means providing a subset of the following device capabilities:

- An ability to advertise its presence to the network,
- Automatic discovery of other devices in the network, locally and remotely,
- Ability to describe its capabilities as well as understand the capabilities of other devices, and
- Seamless inter-operability with other devices.

Among the well-known device coordination standards coming from industry are Universal Plug and Play (UPnP), Device profile for devices (DPWS), Jini and Salutation. In this section, we present only UPnP. We consider that other device coordination technologies will be similar. The aim is to show how devices can communicate with each other.

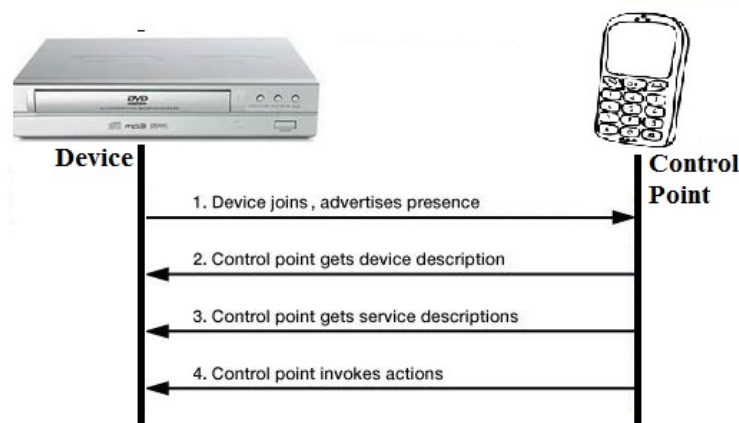


Figure 3.14: A basic mechanism of information exchange between a UPnP device (Media Renderer) and a UPnP Control Point (mobile phone) - Retrieving Device and Service Descriptions

UPnP is independent of any particular technology such as operating systems, programming languages, or even communication medium. At present, UPnP is only being developed and implemented for home entertainment systems, but development is underway to extend UPnP to include industrial machines (devices) as well. It includes the industrial area such as equipment in power plant controls, entire manufacturing plant processes, large transportation systems, etc.

Devices that implement UPnP specifications can act as a device or a control point. A control point has a capability to control other devices but not necessarily to provide a service. Figure 3.14 is an illustration of the basic idea of a UPnP control point discovers a device's capabilities.

As it shown in the Figure 3.14, there are 4 steps of how a UPnP control point controls a UPnP device. Before a control point can invoke a service it must have necessary information in the service description, by listen to the presence advertisements. When a device first comes online, it announces its presence to the network by a broadcast message. Control points can then address the device to obtain a description of the services, determine its current status, and transmit commands to control its behavior.

The design, implementation, and deployment of embedded services are a challenge, in particularly for small devices that have limited capabilities and often different configurations. Several researches on methodologies of software development of embedded systems have been done. For example, ARTEMIS [84] that was initiated by the European Union, is one of the research and development program that focuses on design methods and tools for embedded systems. This includes languages and development methodologies. In [56], reusable building blocks are used to construct embedded applications. A dedicated building block provides a mechanism to access platform-specific functionality of Sun SPOTs devices. The building blocks can be used in combination with other blocks realizing other functionalities such as communication protocols.

With regard to languages for the development of embedded systems, an alternative approach is to implement software functionality is modular programs written in standard, more widely used programming languages. Several research projects and proposals on languages for the development of embedded systems have been done. Esterel [7] is an example of them. SysML from OMG is another example of such languages.

3.4 Software Product Lines

Software product lines (SPL) refer to engineering techniques for creating a portfolio of similar software systems from a shared set of software assets using a common means of production. Manufacturers have long used analogous engineering techniques to create a product line of similar products using a common factory that assembles and configures parts designed to be reused across the varying products in the product line. For example, automotive manufacturers can now create tens of thousands of unique variations of one car model using a single pool of carefully architected parts and one factory specifically designed to configure and assemble those parts.

Similar to the concept of component-orientation, SPL emphasizes software reuse. The characteristic that distinguishes software product lines from the reuse concept of previous efforts is predictive versus opportunistic software reuse. Rather than to put general software components into a library in hopes that opportunities for reuse will arise, software product lines only call for software artifacts to be created when reuse is predicted in one or more products in a well-defined product line.

Within the context of small devices, the SPL approach is promising to help developers to manage the complexity of the variability between devices. With regard to embedded software services, SPL approaches enable the development of a group of embedded software services that can be retargeted for different requirement sets by leveraging common device capabilities and configurations.

A common architecture of SPL, product-line architectures (PLAs) was introduced in [61]. Using a PLA, developers can create software architectures that can be rapidly retargeted to the capabilities of different embedded devices. In a service-based application environment, however, the retargeting of a software application to produce a valid variant for a specific device must also happen at run-time. When a service-based application enters a particular situation, such as included services are changed, the service-based application must very quickly adapt and create a variant for the changed services.

In [20] SCV (scope, commonality, and variability) analysis was introduced as guidance for the design of a PLA. SCV captures key characteristics of software product-lines, including their scope, which defines the domains and context of the PLA, the commonalities, which describe the attributes that recur across all members of the family of products, and the variability, which describe the attributes unique to the different members of the family of products.

In fact, managing multiple devices is not a new problem; even a few solutions have been developed over the years. At run-time, the Composite Capabilities/Pref-

erence Profile (CC/PP) [51] technology provides a standard way to manage device capability information. Figure 3.15 shows an example of a run-time service delivery with regard to the difference of device capabilities on the user's side. In this case, different types of mobile device with different device capabilities want to access the same service, on a mobile service provider.

In order to the mobile service provider to be able to customize services and deliver them to the users, a detection mechanism of device type (and its capability information) is required. There are a number of methods for this. An example is to use UAProf that was defined by the Open Mobile Alliance [102] for user agent detection and user's device capabilities detection. Obviously, the run-time solution of services customization and delivery will influence the service development at design-time. SPL is a good solution for it.

3.5 The ISIS Project

Most parts of this section were taken directly from the ISIS proposal [110] and report as they were used as the basis of the thesis work. The ISIS project was done during 2007 - 2010 and collaboration between universities, software companies and telecommunication companies.

The background for the project is the Internet of Things that is hampered by a fragmented world of standards and players in the field. The different and many proprietary standards can easily lock an end user to a solution that may be outdated

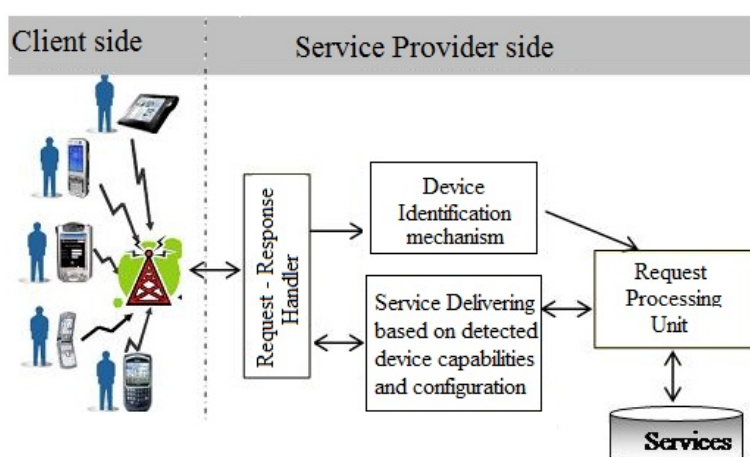


Figure 3.15: The use of device identification for service customization. Different devices need tailored code with respect to its capabilities and configurations.

quickly. It may also be very expensive to add new functionality to these systems. Along with it is the lack of a solution where developers, operators and end users can interact with each other in a fruitful cooperation. The ISIS project has been looking into these challenges, and seeking ways to tackle them by introducing:

1. A user-friendly and special targeted development solution
2. A marked place for components to the Internet of Things
3. A runtime environment where different standards are able to interact with each other
4. An end user composition environment which makes it easy for an end user to create its own services and compositions for the Internet of Things

The real world look of ISIS has been shaped by both ongoing industry trends and readily accessible tools and methodologies that belong to ISIS partners. The innovative part of ISIS is that it has come up with a complete tool chain that supports the design, compose, deploy and execution of services. These tools are well integrated in a seamless manner and they realize an industry process. Along with the tools, a number of good engineering methodologies emerges as a result of extensive user study. We would like to stress that it is the existence of such tools (some of which have good enough quality to be commercialized) and the existence of such engineering methodologies (based on extensive user study) that makes ISIS truly standing out.

ISIS addresses the integration challenges in Internet-of-Things. The main objective of ISIS project is to create a service-driven development process with tools and service execution platforms that substantially improve industrial service engineering from requirements to execution on seamless infrastructures.

The envisaged process Figure 3.16 represents a big step forward in terms of being truly model-driven. Services are identified and modeled based on captured requirements. Service Models are treated as "*first class citizens*", underpinning the whole process from requirements to Service Execution. Supported by the tools, design models based on communicating state machines are derived from the service models. Code generators for the selected execution platform(s) then produce executable code, ready for immediate deployment onto the service execution environment(s). The target execution platform, Service Frame, enables rich and responsive high performance services on a wide range of distributed platforms from mobile units to application servers.

ServiceFrame [63] is a service execution framework developed by Ericsson and commercialized by TellU in order to support convergent services in general. In ad-

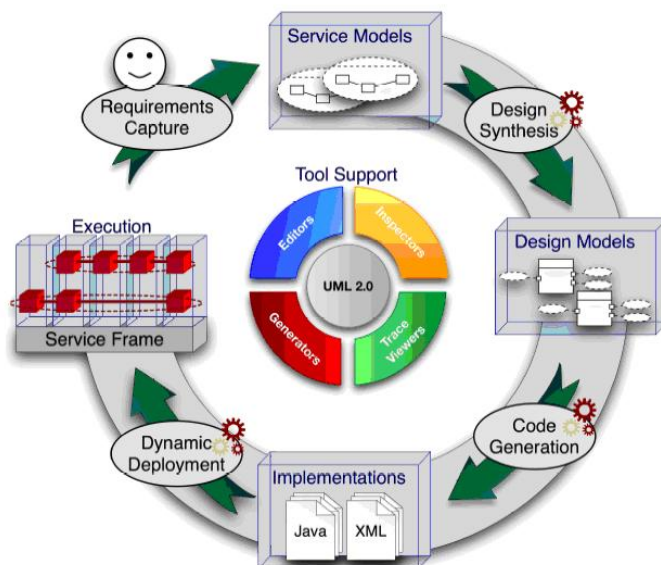


Figure 3.16: The service-driven development process that has been used in the ISIS project.

dition to provide runtime support for communicating state machines it has architectural support for stable domain concepts such as users and terminals, for distributed service execution and for service composition. ServiceFrame runs on a range of platforms from small mobile devices to large application servers, and allow applications to be transparently distributed among them.

ISIS has chosen home automation as the domain for a proof-of-concept. The reason is that there are many services run on a smart home, for example services to control of doors and window shutters, security and surveillance systems, services to control the multi-media home entertainment systems, services for automatic control of plant watering and pet feeding, and services for automatic scenes for dinners and parties. ISIS also assumes that there are several different embedded devices that are connected to the residential gateway.

ISIS project produced several important results. Two of them are an integrated composition environment (ICE) for end users [110] and NTNU's ARCTIS [53][57]. Both of them are used for composing services. ICE is intended for end user while ARCTIS is intended for service developers. With ICE, end user can compose runtime services. Using ARCTIS, at design-time service developers can specify and analyze new services on model levels, and then generate code automatically from the models for different target platform.

Chapter 4

The PMG-pro Method

This chapter presents the proposed PMG-pro (abstract, model, generate, and provide) method. Section 4.1 presents an overview of PMG-pro. The following sections present steps and parts of the PMG-pro: the presentation and abstraction step in Section 4.2, the service library and the frameworks repository in Section 4.3, the modeling step in Section 4.4 and the code generation step in Section 4.5. At the end of each section, a short summary is presented.

4.1 Overview of PMG-pro

For the development of service-based applications, we propose a new method called PMG-pro (Present, Model, Generate, and provide). The architecture of the PMG-pro is shown in Figure 4.1. The PMG-pro architecture consists of three main parts: the presenter/abstractor that is used at the presenting step, the modeling editor that is used at the modeling step, and the code generator/provider that is used at the generating/providing step. There is also a library that is used to store the generated source code and service models at the presenting step. Each service model (abstract) stored in the library binds to source code (concrete).

As it is indicated in the name, the PMG-pro method has four steps: *presenting*, *modeling*, *generating*, and *providing*. The first step is used to present existing pre-made services graphically that conforms to a specific modeling language. Existing services are run-time services that can be accessed employing their descriptions.

In addition to a graphical presentation, this first step generate also code that implements and that acts as a proxy class between the combined service and the concrete services in the service provider. This kind of re-engineering process is possible since the services are often described using open and standard description technologies, for example UPnP, DPWS, WSDL, etc. Moreover, several frame-

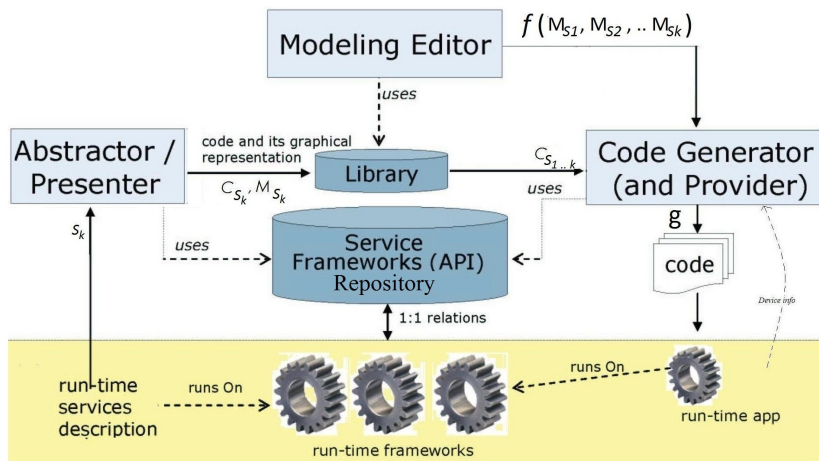


Figure 4.1: The PMG-pro architecture. It consists of the presentation and abstraction step, the modeling step, and the code generation and providing step.

works/API for these technologies are also already available, such as UPnP Cyberlink for Java [52] for UPnP, Ws4d [115] for DPWS and Axis (WSDL2Java) [36] for Web Services. After we presented all existing pre-made services and stored them in a service library, the modeling step of new service-based applications can be started. The new application would be specified as interactions of abstract services. Depending on the modeling language, different symbols and notations can be used to present these interactions. We can use for example activity diagrams or sequence diagrams to specify the interactions. Obviously, the interactions from one abstract service to another would mean service invocations.

The last step is used to automatically generate code that conforms to a specific programming language. Of course the language must be the same with the one used in the presenting step. The code should be generated automatically. A native compiler can be used to compile the generated code. In the next sections, we will present the three steps of the PMG-pro method. We will start with the presentation step, the modeling and code generation step at the end.

4.2 The Service Presentation and Abstraction Step

In this section we present the first step of the PMG-pro that is used to present pre-made services into graphical service representations. For this we need a service presenter and a service abstractor. The architecture of the service presenter and abstractor is shown in Figure 4.2.

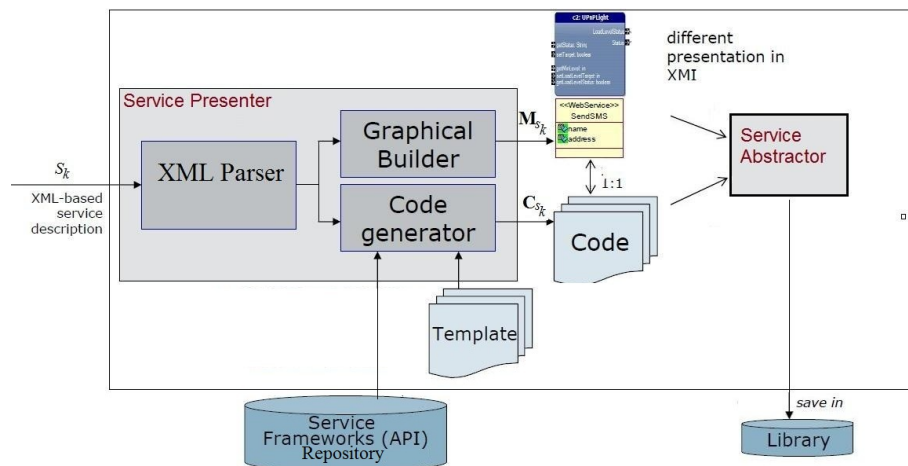


Figure 4.2: The service presenter and abstractor of the PMG-pro. The presents and abstracts services so that service integrators can compose them in a model-driven environment.

4.2.1 Overview

For the development of a service-based application, defining a model of the service-based application (which is a composite service) is only possible if the models of the included (i.e. existing) services are in place. The first part of the PMG-pro is the service Abstractor/Presenter. The abstractor/presenter is used to generate code and a graphical representation of the services. It presents the real services as graphical presentation. This step might be a kind of abstraction and/or presentation step. The difference between them is that the presentation is only a process that transforms one model to another model without reducing the information. In contrast, the abstraction will reduce some information on the target model. The abstraction will always include the presentation process.

In PMG-pro, the presentation consists of a transformation mechanism that transforms service descriptions into graphical representations and source code. It involves re-engineering or reverse engineering processes. The source code is used as a proxy for service invocations to the concrete services that reside in the service providers. For the transformation process, different existing tools and framework can be used. For example, in the context of Web services, we can use WSDL2Java [36] to re-engineer service description in WSDL into source code (Java classes). In order to be a service model, what we need is then visualizing this source code using notations conforming to modeling languages and binding it to the source code.

The service presentation and abstraction step consists of an XML parser, a Code Generator and a Graphical Builder. They are used to handle the XML service de-

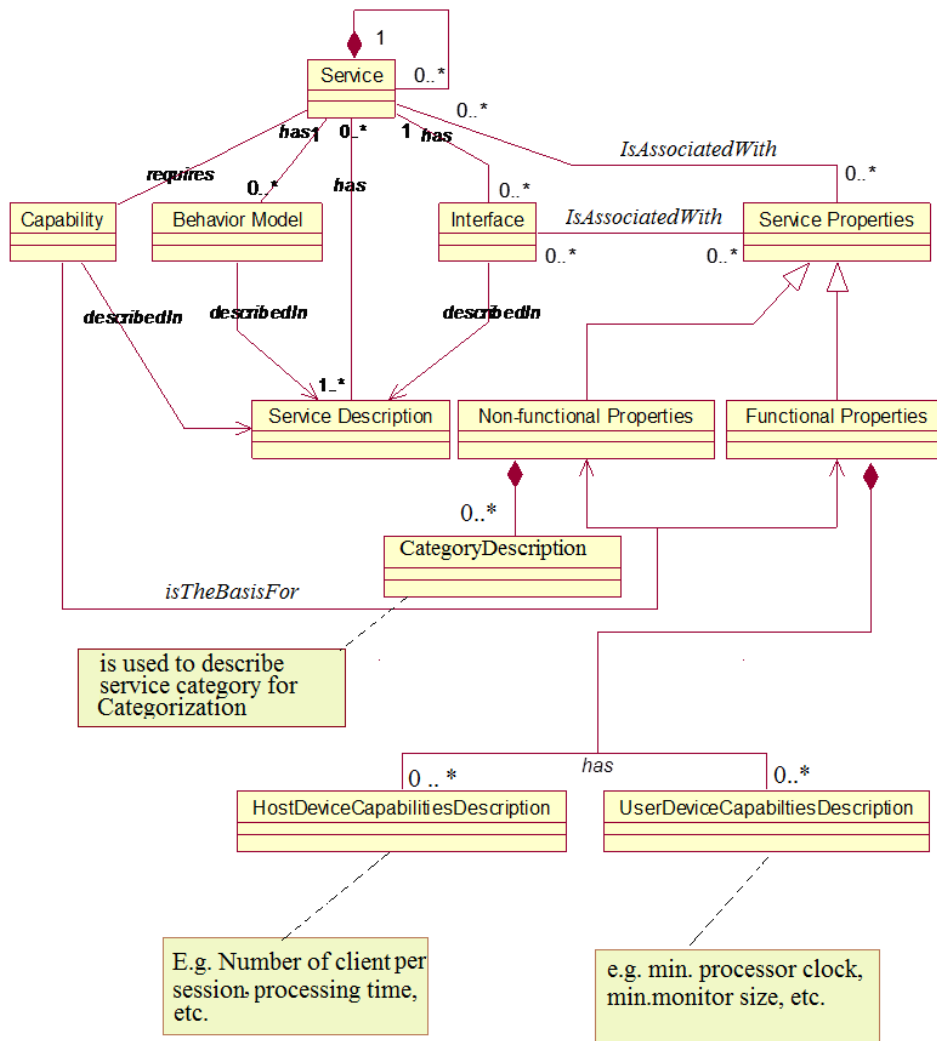


Figure 4.3: A simplified model of service descriptions in the PMG-pro. The model is adapted from the conceptual model of services presented in [103] and [47]. All service properties are described in the service description.

scription, XMI and code. Based on the target modeling languages, the graphical presentation of a service can be different. For example, if we use UML as a modeling language then the service can be presented as a class. The class is presented in XMI as a standard exchange between UML tools. The transformation process from XML service description to XMI is a presentation process and not an abstraction process since they contain the same information. Moreover, the process does not reduce the information.

4.2.2 Service Descriptions

Services can be concrete and abstract. A run-time service is a concrete service. Information about a run-time service is given in a service description (S_k). Thus, a service description is an abstract service since it describes the service. The actual service resides on the service provider.

The fact that the concept of service-orientation systems can be defined, implemented, and described in several ways has introduced different types of implementation of the concept. For example, there exist Web services, OSGi services, UPnP services, DPWS services, Jini services, and other implementations. Obviously, different ways can be used to describe the service functionalities. PMG-pro uses the XML-based service descriptions, which is a service that is described using XML. We consider that XML-based service description is the best one as it is a platform independent. Figure 4.3 shows a conceptual model of service description that is used in this thesis.

From Figure 4.3 it can be seen that a service has one or more service description. We propose a category description and capability descriptions. The category description enables service categorization in the abstraction process. The capability descriptions give information to the service user about the capabilities of services. It can be the capability of service on the service provider or the minimum requirement of capability on the client side to run the services.

4.2.3 The Service Presenter

The development process of service-based applications requires the description of the existing services (service models). However, it is not necessary to have complete service models that describe the structure and the behavior of the existing services. For example, in Web service technology, the Operation descriptions are the important part of the WSDL to compose Web services. The operations describe actions (functionality) supported by the service. The actual behavior of the services

that reside in service providers (the actions) are not necessary known. For this reason, PMG-pro only presents the structure and not the behavior part. The structure is connected to the real implementation of the service (code). The code itself is only proxy to the actual service implementation that resides in the service provider. Their static description is enough information to be able to compose them. Thus, a service can be seen as an object that abstract the details behaviors, but describe the description of the object properties.

The service presentation step is done based on the concept of text-to-model transformation. For example, in the case of transforming UPnP service description into UML class, the presentation involves the transformation of XML (text) into UML class which is stored in XMI. The main consideration is that a service can be presented using any graphical notations/symbols conforming to a specific modeling language. From a service description (s_k), the abstractor/presenter generates a graphical service model (M_{s_k}) conforming to a selected modeling language and source code (C_{s_k}) conforming to a selected programming language. Depending on the selected modeling language, different graphical representations (i.e., notations) can be used to represent the existing services. For example, UML classes, CORBA components, Participants in SoaML, or SCA components are among them. On the system levels, for example SysML [81], a UML profile for system modeling, a building block can be used to represent an existing service.

It must be noted that the service graphical representation must relate to the real services which in this case is done through the code that implements the service invocations. Therefore, it is important to keep the connection (bindings) between

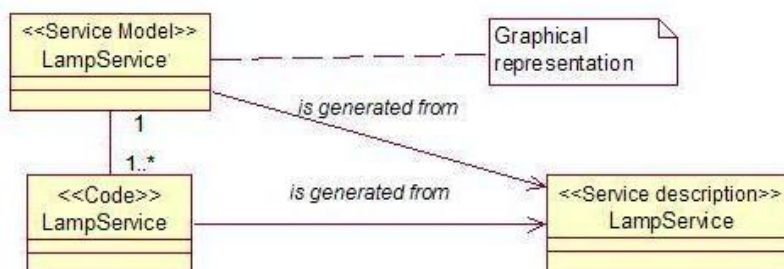


Figure 4.4: The relation between a service description, its model (graphical representation) and source code. The concrete service itself is available at run-time. Services users do not know how the service has been implemented.

graphical representations (i.e. service models) and source code (i.e. implementation for the service invocations). Figure 4.4 shows the relation between a service description, its model, and source code. The concrete service itself (the implementation) is invisible from the service user's point of view.

From the Figure 4.4 it can be seen that there is a 1 : 1 . . * relation between graphical representation and source code which means that a graphical representation of a service may have several variants of source code for implementing the service invocations. They may be implemented using different programming languages on different platforms.

4.2.4 Examples

In a model-driven environment, for the development process of service-based application using existing services, again, it is not necessary to have a complete service model that describes the structure and behavior of the services. For this reason, PMG-pro only presents the structure and not the behavior part. But they can be accessed through ports. The structure is connected to the real implementation of the service (code). The code itself is only proxy to the actual service implementation that resides in the service provider.

Different notations conforming to existing modeling languages can be used to present/model a service. In this section we present four examples showing the use of different modeling languages to describe (to model) a UPnP service. For these examples, a UPnP Light service is used. See Appendix B for the XML service description of the Intel's UPnP Light service.

The first example is presenting the UPnP Light service using an ARCTIS building block [53][57]. This is a transformation of XML-UPnP into XMI-ARCTIS

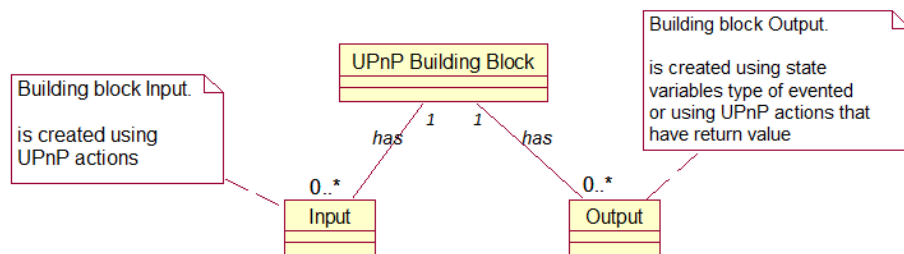


Figure 4.5: A simple building block model to represent a UPnP device (and its service). A UPnP device has inputs (UPnP actions) and generates events (UPnP state variables). A meta-model of UPnP devices is presented in Figure 3.5

building blocks. Before we present the transformation process, a conceptual model of UPnP building block is constructed, see Figure 4.5. This conceptual UPnP building block is created based on the UPnP conceptual model presented in Figure 3.5. As it shown in the figure, a general building block of a UPnP device (and its service) has inputs and outputs. The UPnP actions can be used as inputs of building block. For the outputs of building block, we can use the UPnP state variables. In UPnP, a notification of changed state variables (type of evented) will be sent to others UPnP devices that subscribe the services. With this, we use the outputs as events. Another type of output is constructed by using the UPnP actions that have return value. This type of output generates output when the UPnP action is invoked.

Using ARCTIS, the conceptual model of UPnP building block presented in Figure 4.5 can be implemented. ARCTIS is a modeling editor that is based on UML. More than that ARCTIS is supporting a model-based engineering technique for reactive systems [66]. Software developers can use ARCTIS to specify, analyze, and verify software system using models [57]. ARCTIS has also model checking supporting for automated model correctness. Instead of UML classes ARCTIS uses building blocks as a basic entity. An ARCTIS building block is considered as a kind of activities encapsulation which can be accessed through its ports. In this case, one building block can be used to represent one service. A service-based application in ARCTIS is then can be specified as a building block diagram. A service-based application can be, again, encapsulated and presented as a new building block. Obviously, this new building block can be used to build a new building block diagram as new specification of a software system. With this, an incremental development of a large service-based application can be achieved. About ARCTIS, interested readers should refer to [65, 53, 57].

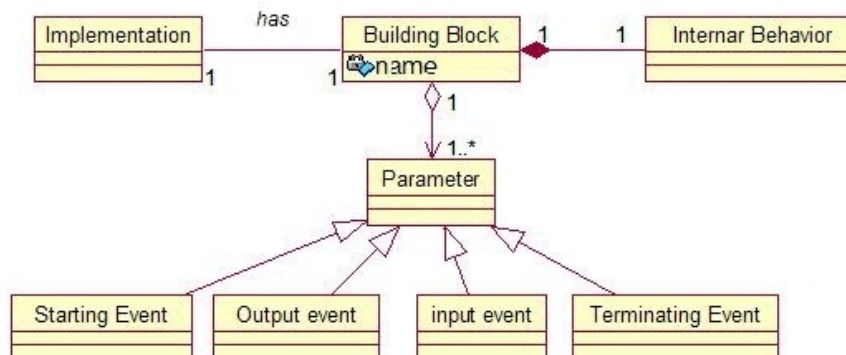


Figure 4.6: A conceptual model of ARCTIS building block - A basic element of model in ARCTIS [53, 57, 65].

To be able to present a UPnP light service using an ARCTIS building block, we need to transform the XML-UPnP service description into XMI data (the XML format used in ARCTIS). The XML stands for eXensible Markup Language. Thus, the XML is a language. By considering this, an XML UPnP service description is a model since the description is an instance of the language. Similarly, an ARCTIS building block is also a model as it is an instance of ARCTIS that is based on UML 2.0 [53, 57, 65]. With these facts, the theory of model transformation presented in Section 3.2.4 can be applied.

To automate the transformation process, the concept of model-to-model transformation can be applied. For this, it requires the conceptual models of the source and target modeling languages. These conceptual models are used to construct the transformation rules. Figure 3.5 shows the conceptual model of the UPnP devices and its services, while Figure 4.6 shows the conceptual models of ARCTIS building block in a UML class diagram.

Using the UPnP conceptual model (Figure 3.5), the ARCTIS conceptual model (Figure 4.6) and the conceptual model of UPnP building block (Figure 4.5), transformation rules for transforming an UPnP service description into a UPnP-ARCTIS building block can be constructed as presented in Table 4.1.

Table 4.1: Transformation rules between UPnP - ARCTIS Building Block

UPnP Device	ARCTIS Building Block
Device name	Building block name
Action	Input event and output event ^a
Action argument	Type of input event
State variable	Output event
Type of state variable	Type of output event

^aOutput event for actions that have return value

Using the constructed transformation rules an UPnP service description can be transformed into ARCTIS building block automatically. Figure 4.7 shows a representation of a UPnP service in a ARCTIS building block as a result of the transformation process. It must be noted that the process is only for visually representing existing services. It can use graphical or textual representation. The internal behavior of the service is invisible.

Similar with the transformation process of UPnP-ARCTIS above, the UPnP Light service above can be presented using a SCA Component, a UML class diagram, a SoaML Participant that are shown in Figure 4.8, Figure 4.9 and Figure

4.10, respectively. It can be seen that the presentation process is only done for the purpose of visualization of existing services. The service is not static, but the description is static. UML classes, for example, can represent static entities with dynamic behavior. It is handled in a static way during composition at runtime and design time.



Figure 4.7: An ARCTIS building block is used to visually represent the UPnP Light service

In these examples we also consider that all the service models are connected to the same source code for the service invocations. Listing 5.3 shows a Java source code for the invocations of the UPnP Light service. In PMG-pro the source is generated automatically during the abstraction/presentation step. A complete source code is presented in Appendix D.

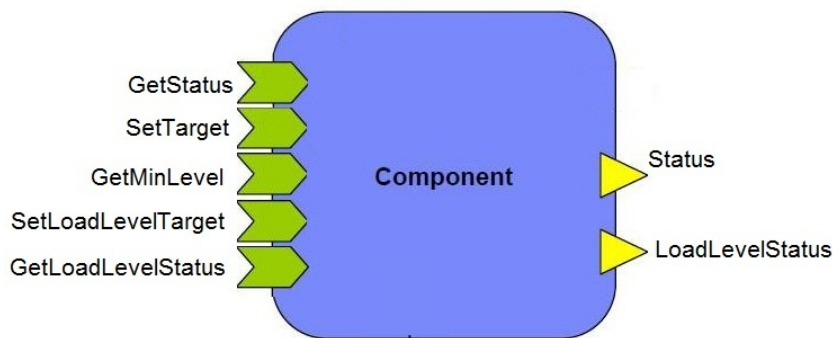


Figure 4.8: A SCA component is used to represent the UPnP light service.

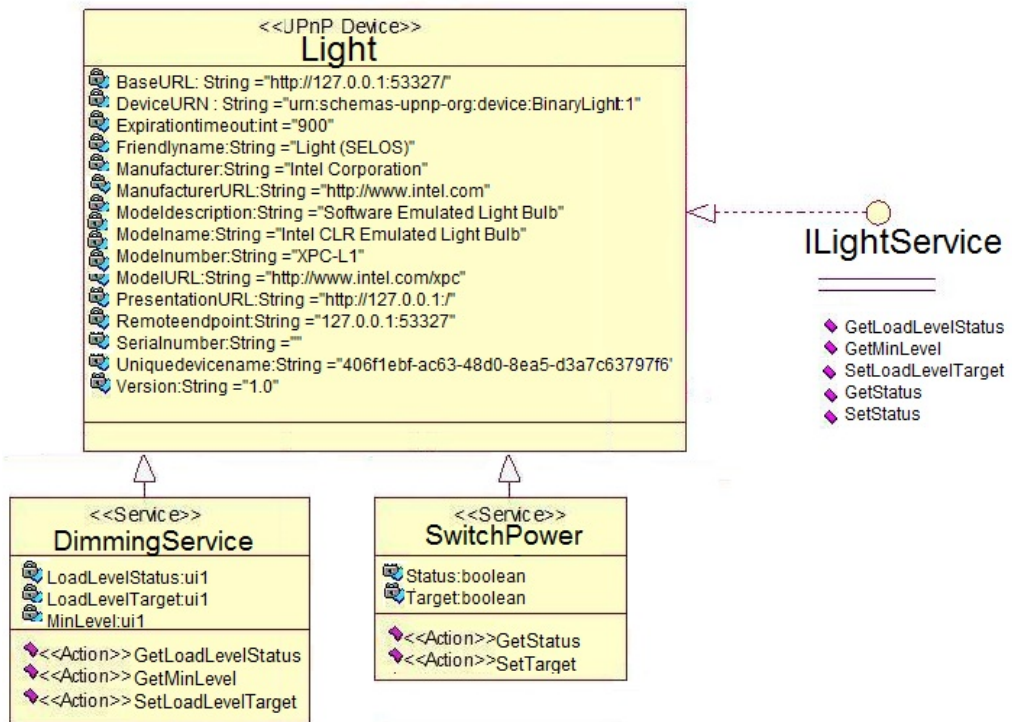


Figure 4.9: UML classes are used to represent the UPnP Light service.

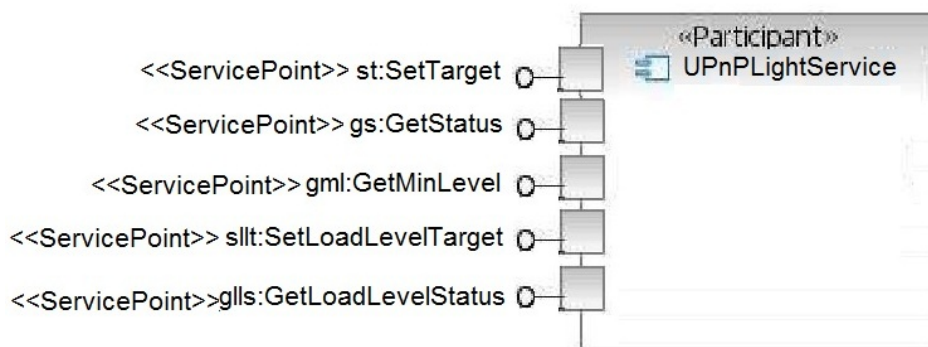


Figure 4.10: A SoAML participant is used to represent UPnP Light service.

4.2.5 The Service Abstractor

A service model can appear in different abstraction levels. For example, end-users may use an icon to model a service, while a developer may use a class to model a service. In PMG-pro, a service abstractor is used to abstract services at different abstraction levels.

4.2.5.1 A Common Service Model

Existing run-time services might have been implemented using different standards and technologies. Consequently, they use different ways of describing services, even the services with similar functionalities. In PMG-pro, from a service description a pair of code and its graphical representation of the service are generated. For similar services, there will be a common service model. An example of a common service model, which is a platform independent model, is illustrated in Figure 4.11.

In Figure 4.11, three Lamp services that provide the same functionalities are shown. The UPnP_Lamp service and the UPnP_Light use UPnP schema to describe

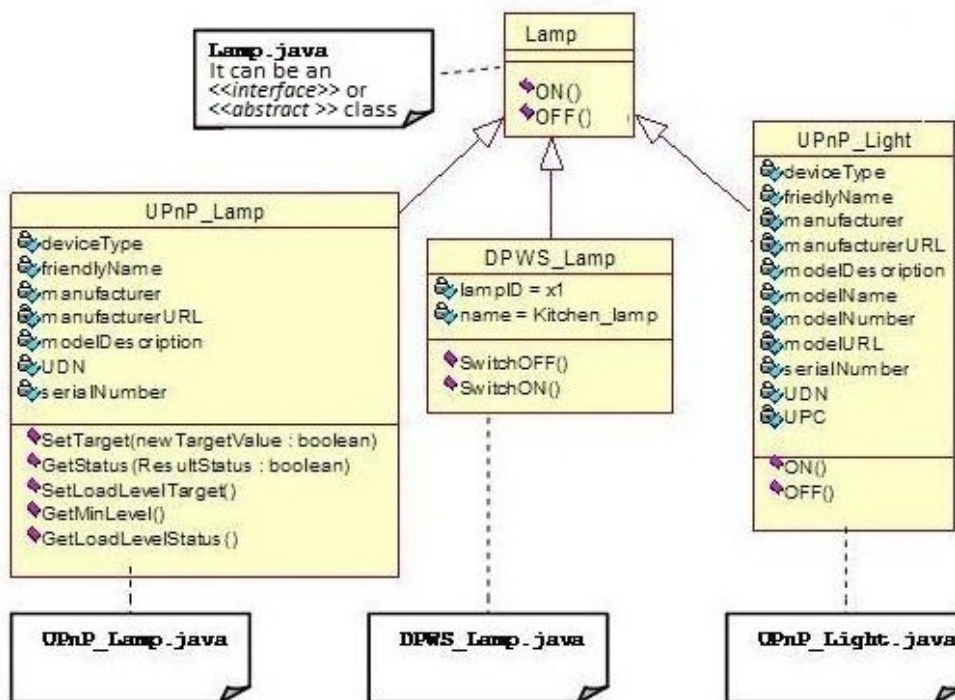


Figure 4.11: A platform independent service models and its code. In this example, a Lamp is a common model of different implementation of services: UPnP_Lamp, DPWS_Lamp and UPnP_Light services.

services while the DPWS_Lamp uses DPWS to describe the service. All the Lamp services provide functionalities to switch ON-OFF the lamp even though they use different operation names. From these different service models a Lamp service which has ON and OFF functionalities can be defined.

All the service models are connected to its code, and so code for the Lamp service model. Obviously, the code may vary in terms of programming languages. In the Java context, we can use an interface or an abstract class to declare the code. Listing 4.1 shows an example of the code for a common service model of the Lamp service, when an abstract class is used to declare the Lamp code. It can be seen that the abstractLamp class is an abstract class with two abstract methods. Later, in the code generation step of PMG-pro when the class is used in a model, implementation class of the class will be generated automatically based on a template.

Listing 4.1: Lamp Code

```
public abstract class abstractLamp {
    ...
    public void abstractLamp(){
    }

    public abstract void ON (String selectedService);

    public abstract void OFF (String selectedService);

    .....
}
```

It must be noted that the generalization is one way up. The generalization process is more like a classification process, where similar service models will have one general class. Subclasses will not be modified. However, to show the case we present an example of the modified UPnP_Lamp class that is shown in Listing 4.2. The information about super and subclasses is stored in the Service Library.

Listing 4.2: Modified UPnP_Lamp Code

```
public class UPnP.Lamp extends Lamp{

    public void setTarget(boolean newTargetValue){
        // action code
    }
    .....
}
```

4.2.6 Summary

In PMG-pro, the presentation step is used to present services into graphical representation and code implementing service invocations. We consider that XML is a language therefore XML UPnP service description is a model since it is an instance of XML. Transforming the model into another models for example UML class can employing the concept of model-to-model transformation (M2M).

By developing different abstractor/presenter different notations conforming to modeling languages can be chosen. In addition to graphical service representation, the presentation step generates source code conforming to a selected programming language. For this, the concept of model-to-text (M2T) transformation is used. The service graphical representations are used by service developers to model the new service-based application while the code is used to generate code in the code generation step.

4.3 The Service Library and the Service Frameworks (APIs) Repository

The Service Library and the Service Frameworks (APIs) Repository is the central structure of the PMG. Three of other parts of the PMG-pro (the service abstractor, the model editor and the code generator) are connected this part.

The Service Frameworks (APIs) Repository contains service frameworks and API's that are used by the service abstractor as a foundation for the transformation of service descriptions into notations conforming to a modeling language (for example, UML classes) and to generate implementation class. Examples of existing service frameworks are [52] for UPnP and [36] for Web Services. Other frameworks for embedded systems may also be included in the service repository and library.

The service library contains service models and its connected code. The service library is used by the service integrator. By the use of a model editor, a service integrator use the stored graphical service models in the service library to specify service-based application, while the code generator use the service library for the purpose of code generation from a service-based application model. We will present the code generation method later, in the code generation step.

The service library is also used by service abstractor to store service graphical representation and its code in a hierarchical manner. It is built based on ontology. One important assumption that we use to build the service hierarchy is that, the service description must contain enough information for a service categorization.

Figure 4.12 shows an example of service taxonomy in a smart home domain, where different implementation of `MediaRenderer` services (e.g. UPnP and DPWS) are categorized under the `MediaRenderer` services, `Entertainment Services`, and `Smart homes services`, respectively. All of the `MediaRenderer` services at level 0 of hierarchy have graphical representation (with `<<PSM>>` stereotype). A `MediaRenderer` services at level 1 of hierarchy is defined as a common model to represent all the media renderer services. It should be noted that the code will be used by the code generator and the models will be used by modelers to specify new service-based applications.

At the level 1 and above, there is a similar relation between model and code. The source code binds to the service models at level 1 and above is not a real code. At a very high level of hierarchy, a service model is connected to abstract information about possible target platforms (source code) that are available and used on the code generator. Based on the selected target platform, the code generator will generate tailored source code.

In smart home domains, ontology has been used for composing services such as used in [111, 87]. In [87] the use of ontologies enable the effective description of the heterogeneous services and resources residing in the smart homes. It includes

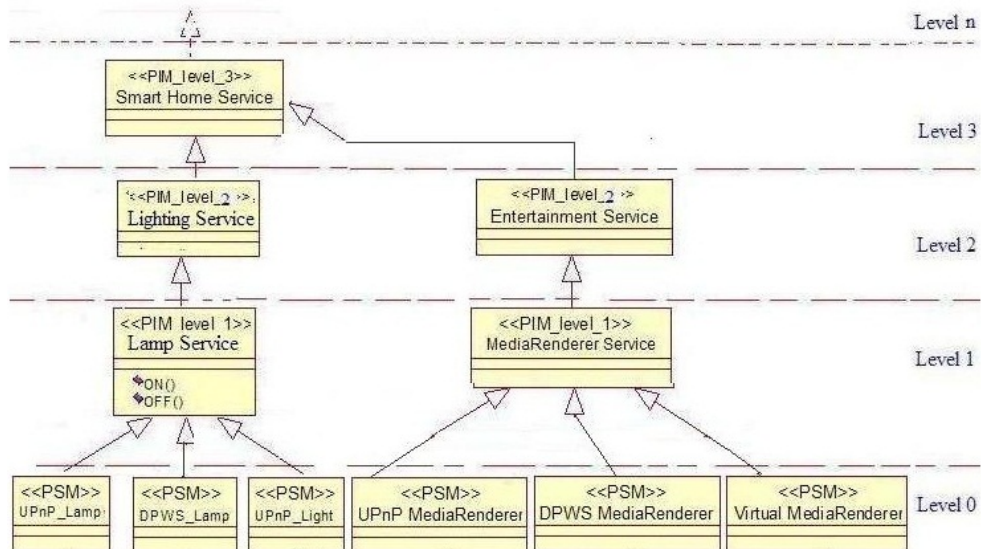


Figure 4.12: An example of taxonomy of service models. Each model is connected to source code. The source code itself can be at any programming language.

a proposal for an automatic ontology-based service discovery and dynamic service composition.

Referring to the idea of service taxonomy presented in Figure 4.12 a smart home service ontology is created. A basic taxonomic tree of the General Formal Ontology showing in Figure 4.13. This ontology can be used to automate the construction of service hierarchy in PMG-pro. In this case, PMG-pro assumes that all the devices in smart homes provide embedded services.

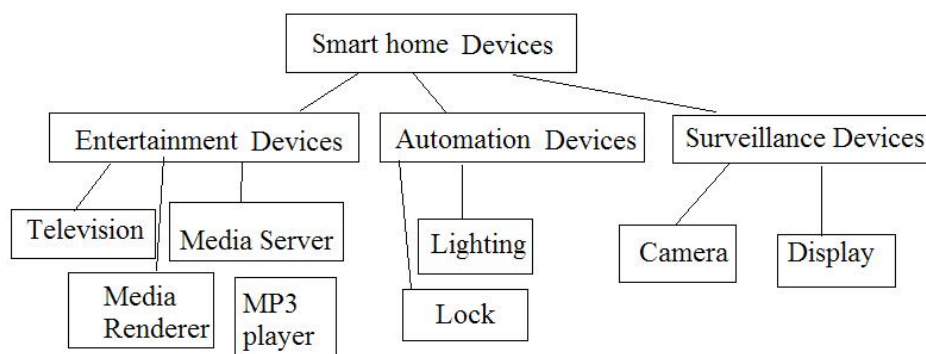


Figure 4.13: A General formal ontology of smart homes devices used in PMG-pro.

In PMG-pro Figure 4.13, smart home devices are categorized into three classes; entertainment, automation and surveillance devices. Including in the entertainment devices are televisions, mp3 players, media servers and media renderers devices. These devices provide functionality that can be discovered and used.

4.4 The Modeling Step

PMG-pro is a language independent method. It is possible to use different existing modeling languages and different modeling editors. The requirement is that the presenter must generate notations (i.e., abstract service models) that conform to the chosen modeling language. Using the generated service models, service integrators can model new service-based applications.

There is no direct contribution of PMG-pro in the modeling step. The only contribution is a possibility of using different modeling languages to model service-based application. However, in this section we describe how service-based applica-

tions can be modeled in PMG-pro. We focus only activity and sequence diagrams. But obviously, it depends on the chosen modeling language.

4.4.1 Overview

In software development, models are used to describe the structures and the behaviors of the software systems. The structure specifies what the instances of the model are; it identifies the meaningful components of the model construct and relates them to each other. The behavioral model emphasizes the dynamic behavior of a system, which in UML for example, can essentially be expressed in three different types of behavioral models; interaction diagrams, activity diagrams, and state machine diagrams.

4.4.2 Collaboration Activities Diagram

Even though for model-driven engineering, a state-machines diagram is considered as the most executable models [35], we are still interested in using UML activity diagram and collaboration diagram. The reason is that, from activity diagrams we can generate state machine diagram [66]. The UML activity diagrams are used mostly to model data/object flow systems that are a common pattern in service-

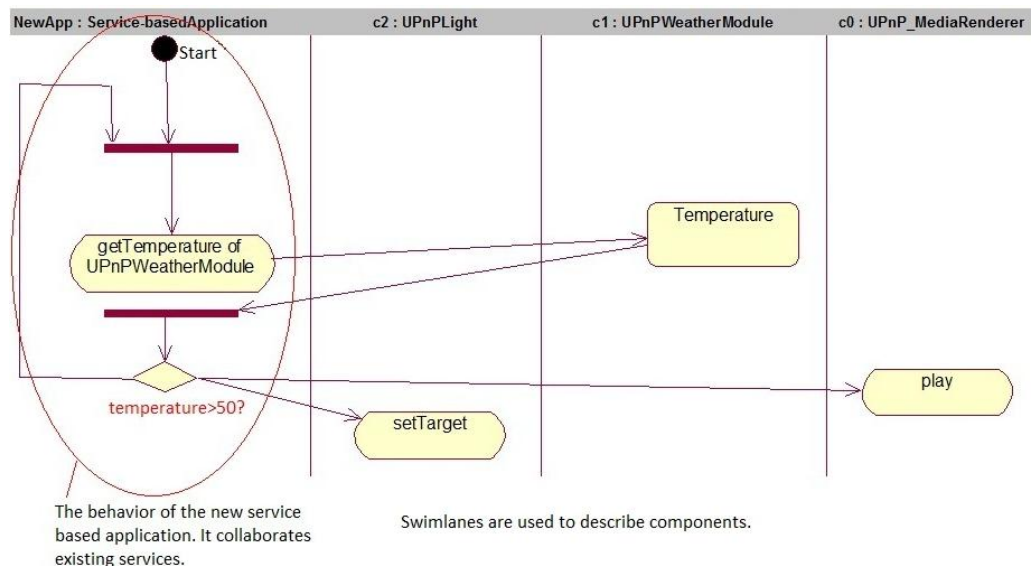


Figure 4.14: A service-based application model is specified using an activity diagram. The figure shows a service-based application that uses three existing services. Each service has internal activities. A swimlane is representing a service

based application models. The activity diagram is also good to model the behavior of systems which do not greatly depend on external events. A system that is mostly having steps that run to completion, rather than being interrupted by events and requiring object/data flow between steps.

Thus, a model of service-based application in PMG-pro can be presented as a collaboration activity where connections between service models are defined using activity nodes. See Figure 4.14 for an example. The figure shows a service-based application that uses three existing services. Each of the service has its own internal activities. From the view of the service-based application, the internal activities are seen as a simple service.

4.4.3 Sequence Diagrams

There are several ways to model interactions (connections) between services (i.e., service composition) in a service-based application diagram. For a simple service composition, a sequence diagram is simpler than an activity diagram. By simple composition we mean a composition that does not have dataflow and without any activity between service models.

By developing a service abstractor/presenter that is able to generate a UML class from a service description, we can then use the objects of the class to specify new service-based applications in form of sequence diagrams. Figure 4.15 shows an example of a service-based application specified in a sequence diagram.

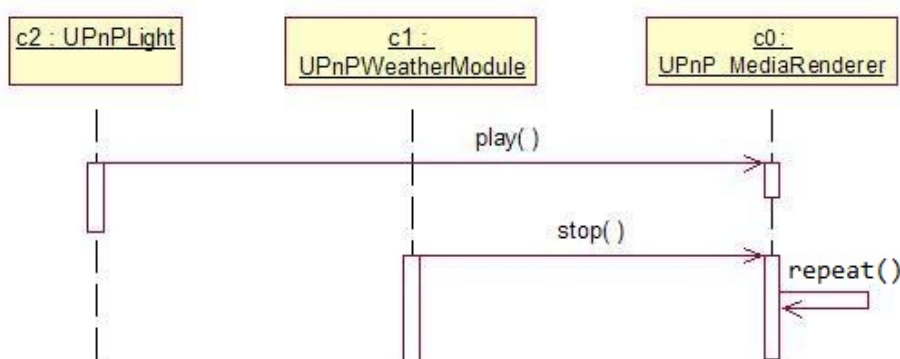


Figure 4.15: A service-based application model specified is using a sequence diagram. In this figure, three different UPnP services are used to build a service-based application. The application will play music when a UPnP Light service is invoked, and will stop it on a certain value of weather parameter.

4.4.4 High-level Models

Using service hierarchy (see Figure 4.12), it would be possible for the modelers to model new service-based applications at any levels of hierarchy. The higher the level of hierarchy the more platform-independent the models would be. By high-level models, we mean service-based application models that are built using platform-independent service models at level 1 and above of service hierarchy (see Figure 4.12).

High-level models might use presentations that are close to the end-users, for example pictures that describe the real services. With regard to the service hierarchy (see Figure 4.12), for end-users, the services may be presented in the highest level of service hierarchy. Although this topic is out of the PMG-pro context, a short explanation of how the idea can be done is given. In the ISIS project [110], ICE, an end-user composition tool that enables end-users to compose different services at run time was developed. We consider that by developing a specific service presenter and abstractor, service presentations (service models) for end-users and code for implementing the service models can be generated. The service models can then be used by the end-users to model the composition.

4.4.5 Model Correctness

One of the major issues when using activity diagrams and sequence diagrams relates to semantics. With regard to the UML activity diagrams, several works have been to tackle this issues [29, 12, 107]. Several proposals have been also proposed, for example [59, 18, 26].

However, PMG-pro is language independent. It uses existing modeling languages to model a service-based application. With this, the semantics of the models and model correctness follow the chosen modeling languages. For example, when we use ARCTIS to model a service-based application then the model correctness is checked using its built-in model checker. Accordingly, if we use a UML tool to model service-based applications then the semantic and the model checking mechanism follow its built-in model checker if they have. This leads to a conclusion that the semantics of activity and sequence diagrams are not a big problem in PMG-pro, while the semantic of the structure part is easy.

4.4.6 Summary

Modeling of service-based applications in PMG-pro is restricted to the service models stored in the service library. The stored service models are a representation

of existing run-time services. Different modeling languages can be used to model service-based applications. PMG-pro present run-time services using notations that conforms to the chosen modeling languages.

Since PMG-pro uses existing modeling languages, the model correctness depends on its built-in model checker. PMG-pro assumes that all modeling languages have its own (built-in) model checkers.

4.5 The Code Generation and Providing Step

To execute the models of service-based applications that have been produced in the modeling step, PMG-pro applies code generation instead of model interpretation. We apply model to a text transformation approach. The benefit of this approach is that we have an additional checker for correctness of model since the approaches use existing programming languages.

Many modeling tools, especially domain-specific modeling tools, come with their own code generator. In this case, PMG-pro can use those code generators. However, for the purpose of proof-of-concept where a service-based application will be provided as a new service, a specific code generator must be developed. In this section we present a concept of PMG-pro generates code from a model of a service-based application.

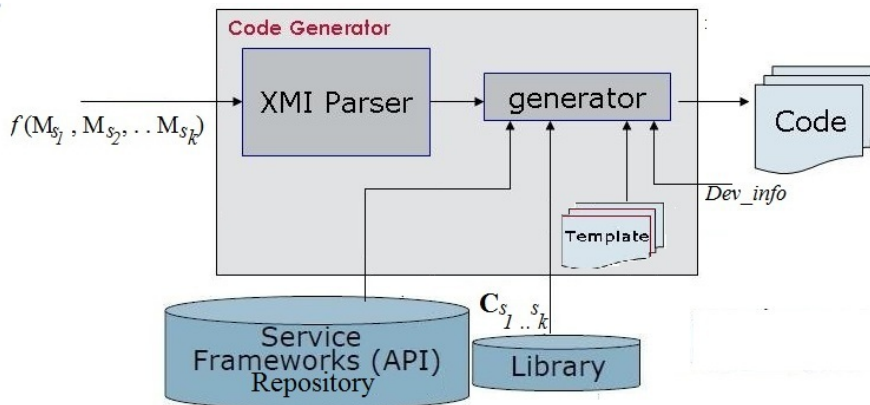


Figure 4.16: The code generator. A template is used to generate source code.

4.5.1 Overview

Figure 4.16 shows the architecture of the code generator. As it can be seen in the figure, the input for the code generation of PMG-pro is a model of the service-based

application that can be an activity diagram or a sequence diagram. The XMI parser reads a model of the service-based application that is presented in a XMI file. The output of the parser is given to the generator.

4.5.2 Code Generations

Models have two parts; structures and behaviors. In PMG-pro the basic structure is described using static class and building blocks while the behaviors are expressed using UML activity nodes and actions in sequence diagrams.

Code is directly interpreted from the activity elements. For example, a line connection from one port in a building block to port in another building block means service invocation. Code for this connection is simply copied from proxy classes that contain code for the actual invocations.

4.5.2.1 Code from Activity Diagrams

Models of a service-based application can be defined using activity diagrams. Thus, a code must be generated from this activity diagram. In this section we present a basic idea of how PMG-pro generates code from activity diagrams. Figure 4.17 and listing 4.3 shows an example of A merge node and its code generated code.

A merge node is a control node that brings together multiple alternatives. It is not used to synchronize concurrent flows but to accept one among alternate flows. It has multiple incoming edges and a single outgoing edge. The execution of the activity diagram can be described as shown in listing code 4.3. In [95] it is listed other examples of code generation for other activity nodes. A similar concept can also be found in [10] and [30].

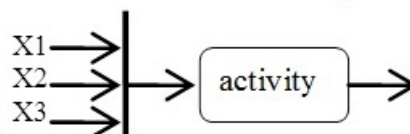


Figure 4.17: A merge node. The figure shows an activities and three input data flows. The activity will be executed all the data flows are arrived.

Listing 4.3: An example of code generation from a merge node

```
void mergeNode(){
if ((X1)&&(X2)&&(X3))
activity (X1,X2,X3);
}
```

With regard to the ARCTIS code generation, PMG-pro code generator applies only a simple transformation process. It reads and converts directly activity nodes (with control and object flows) into code. So, it works only on a simple activity diagram due to the formalism. The reason is that the code generator in PMG-pro is developed only for the purpose of proof-of-concept. However, since the method is independent of languages and tools, PMG-pro can use any built-in code generator. In contrast, ARCTIS converts a collaboration activity diagram into a state machine which is executable. For this, cTLA is used for the reasoning [55]. Code is then, generated from the generated state machines. It works on more complex activity diagrams.

4.5.2.2 Code from Sequence Diagrams

Some systems have simple dynamic behavior that can be expressed in terms of specific sequences of messages between a small, fixed number of objects or components. In such cases sequence diagrams can completely specify the system's behavior. Code can be generated from these simple cases. Listing 4.4 shows an example of generated code from the sequence diagram shown in Figure 4.15.

Listing 4.4: An example of code generation from a sequence diagram

```
void sequenceCode
{
UPnP_MediaRenderer c0;
UPnPWeatherModule c1;
UPnP_Light c2;

c1.c0.play();
c2.c0.stop();
c0.repeat();
}
```

However, in many cases, behavior is more complex, e.g. when the set of communicating objects is large or highly variable, when there are many branch points (e.g. exceptions), when there are complex iterations, or synchronization issues such as resource contention. In such cases, sequence diagrams cannot completely describe the system's behavior, but they can specify typical use cases for the system, small details in its behavior, and simplified overviews of the behavior. Live Sequence Charts (LSCs) is one example of solutions to this problem.

4.5.2.3 Code Generation Method for Handling the Device Capabilities

In the context of MDA, different target platforms require different tailored code. In the context of embedded systems, the terms of device capability and configuration are often used instead of platforms. Device capability and configuration introduce problems for code generation even within the same device capability. For example, it can occur the development of mobile applications. In this case, it can happen that a working source code for one specific mobile phone will not work on another one of the same type. For instance, this can be caused by a difference in the user configurations (e.g. memory size) or/and in the device capabilities (e.g., the resolution of camera).

Listing 4.5: Lamp Code

```
public class Lamp extends abstractLamp{
    ...
    public void Lamp(){
        UPnP_Lamp upnp_lamp= new UPnP_Lamp();
        UPnP_Light upnp_light=new UPnP_Light();
        DPWS_Lamp dpws_lamp = new DPWS_Lamp();
    }

    public void ON (String selectedService){
        if (selectedService.equalsIgnoreCase("UPnP_Lamp")) {
            upnp_lamp.setTarget(true);
        } else if (selectedService.equalsIgnoreCase("UPnP_Light")) {
            upnp_light.ON();
        } else if (selectedService.equalsIgnoreCase("DPWS_Lamp")) {
            dpws_lamp.SwitchON();
        }
    }

    public void OFF (String selectedService){
        if (selectedService.equalsIgnoreCase("UPnP_Lamp")) {
            upnp_lamp.setTarget(false);
        }
        .....
    }
}
```

Services in a service-based application environment are considered being provided by third parties; therefore they can easily come and go. It would be very possible that included services are not available when a service-based application is implemented, deployed, and run. To generate code for a device with a specific device capability, two solutions are proposed. The first solution is generating code for possible aggregated services in the ontology. For example, the abstract `Lamp` class shown in Figure 4.11 can be implemented differently during the code generation process. This solution would also support run-time adaptation in case of services are removed or new services appear. Listing 4.5 shows an example of possible generated code for implementing the `Lamp`, using the first solution.

The `Lamp` class above is an implementation class of the abstract class presented in Listing 4.1. It extends the `abstractLamp` class for implementing the abstract methods. This implementation class is generated automatically based on a template.

The second solution is to generate code only for the present (specific) services. This solution requires information about what devices are available at run-time. This means code cannot be generated before the platform is used, which essentially means on-demand code generation. For example, if the available `Lamp` service is only `UPnP_Lamp`, then the code generator will only generate (use) the `UPnP_Lamp.java`. The code generator will not generate (use) code for other `Lamp` services (i.e., `UPnP_Light` and `DPWS_Lamp`).

4.5.3 Summary

Models describe the structure and the behavior of a service-based application. In PGM-pro, the code from the structure is generated by objects instantiation while code from the behavior is generated by interpretation of the interaction between object instances. We adapt the DSL environment where we assume a service as an implemented concept. For this reason, we do not specify which languages should be used to model service-based application.

Chapter 5

Proof-of-Concept

Parts of the PMG-pro have been prototyped using Java programming language. This chapter presents the prototypes and case studies showing the proof-of-concept. Section 5.1 is an overview of the prototypes. Section 5.2, 5.3, and 5.4 present the prototypes of the service presenter for ARCTIS, the service library, and the code generator. For the proof-of-concept, three case studies are presented in Section 5.5.

5.1 Overview

We have prototyped parts of the PMG-pro to show the idea of presenting visually services using graphical representation. Our plan is to extend the service abstractor/presenter to be able to generate different notations supporting different modeling languages and editors. For this, we developed a general user interface as shown in Figure 5.1. It must be noted that not all the links in the figure are implemented.

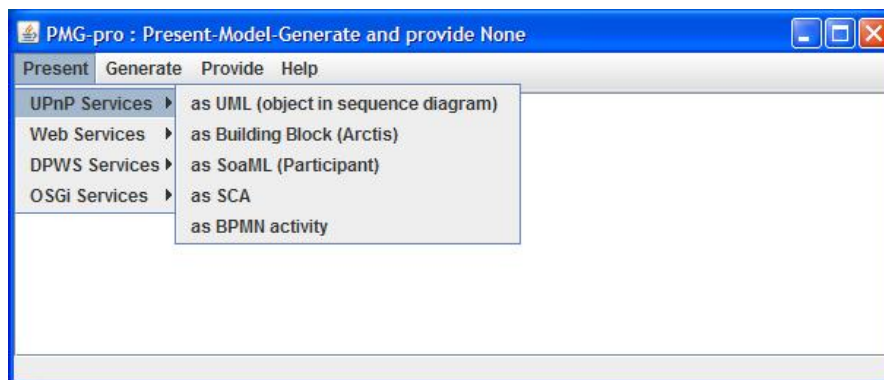


Figure 5.1: The screenshot of the user interface of PMG-pro.

5.2 Service Presenter

The focus of the prototype construction is to show that PMG-pro is worked. For this, we focus only on the automated presentation of run-time services and code generation parts. For the service representation, we have implemented two service presenters. The first service presenter is capable to present run-time services as ARCTIS building blocks. The second service presenter is developed to present run-time services as UML classes in Rational Rose [85].

Using ARCTIS, the behaviors of a service-based application are modeled using collaboration activity diagrams while using UML (Rational Rose), the behaviors are modeled using sequence diagrams. Hence, the PMG-pro prototype supports for two modeling techniques; collaboration activity diagram and sequence diagrams. Obviously, the semantics of the language follows the chosen modeling languages.

The two prototypes of service presenter mentioned above are similar. In this thesis we present only the prototype of the service presenter that implements a transformation of UPnP services into ARCTIS building blocks. For this transformation, we utilized XMI (XML for meta-data interchange). The first prototype is able to transform a UPnP service description (XML) into an ARCTIS building block (XMI 2.1), while the second prototype is able to transform WSDL service description (XML) into UML classes.

The use UML class (which is static) to present service (by nature is dynamic) is possible since the presentation in PMG-pro is only visually presenting (and abstraction) of run-time services. It is not the actual services. The actual services and its behaviors reside in the service providers. In PMG-pro, a service model is an abstract presentation of an existing service at high level. Access functionalities to the actual services are done by the connected code.

5.2.1 UPnP Service Description

UPnP uses XML as a schema language for defining device and service descriptions, control messages, and eventing. A UPnP-based device can implement zero or more services. However, each UPnP device must host an XML device description that contains specific information about the device, information about the services of the device, and even information about the nested devices.

The service description itself is an XML document that lists actions and state variables that apply to a specific service offered by the UPnP device. Figure 5.2 illustrates a power service description (SetTarget) of the UPnP Light device in XML. See appendix B for more details.


```

<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  .....
  <actionList>
    <action>
      <name>GetStatus</name>
      <argumentList>
        <argument>
          <name>ResultStatus</name>
          <direction>out</direction>
          <relatedStateVariable>Status</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SetTarget</name>
      <argumentList>
        <argument>
          <name>newTargetValue</name>
          <direction>in</direction>
          <relatedStateVariable>Target</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    .....
  </serviceStateTable>
</scpd>

```

Figure 5.2: An example of a UPnP service description. The actions are listed in the service description. See Appendix B for more detail.

5.2.2 An ARCTIS Building Block Template

To generate ARCTIS building blocks we use a building block template (an XMI file). We constructed the template based on a simple ARCTIS building block shown in Figure 5.3. The figure shows the internal activity of a simple ARCTIS building block for UPnP services in an activity diagram. With regard to the UPnP services, important parameters of a building block are operations (action), receive signals (EVENT) and variables. Therefore we only consider these parameters on the construction of the ARCTIS building block template.

5.2.3 The Transformation

In general, transformations are defined by mapping rules using a transformation language. Each mapping rule describes what one, or more elements in the source model should be transformed to in the target model. When all mapping rules are applied, the mapping describes the complete transformation from the source model to the target model.

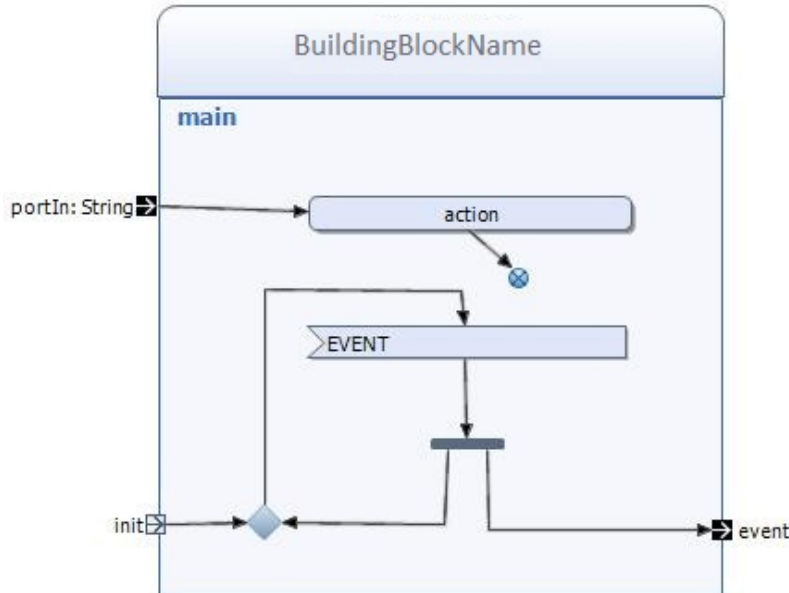


Figure 5.3: The internal activity diagram of a simple ARCTIS building block for UPnP services. The building block has one action (operation) and one event out. This building block is used for the construction the template for a UPnP ARCTIS building block.

```

<?xml version= 1.0 encoding= UTF-8 />
<xmi:XMI xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
  <uml:Package xmi:id="_i0gnMMUvEd6LLbh8D5bTNw" name="test.block">
    <packageElement xmi:type="uml:Activity" xmi:id="_i1Q0EMUvEd6LLbh8D5bT
      <ownedBehavior xmi:type="uml:StateMachine" xmi:id="_jNZBQMUVEd6LLbh8
        <region xmi:id="_jNZBQsUVEd6LLbh8D5bTNw" name="&lt;region>">
          <subvertex xmi:type="uml:Pseudostate" xmi:id="_jNsJQMUVEd6LLbh8C
            <subvertex xmi:type="uml:State" xmi:id="_cb6uAd66EecGjOrFcy00Tg"
              <subvertex xmi:type="uml:Finalstate" xmi:id="_cb6uAt66EecGjOrFcy
                <transition xmi:id="_cbxkEN66EecGjOrFcy00Tg" name="init /" sour
              </region>
            </ownedBehavior>
          <ownedOperation xmi:id="_H1scYN66EecGjOrFcy00Tg" name="action"/>
          <ownedParameter xmi:id="_s0sUN65EecGjOrFcy00Tg" name="init"/>
          <ownedParameter xmi:id="_DF7wIN66EecGjOrFcy00Tg" name="portIn" type=
          <ownedParameter xmi:id="_Ec8zcd66EecGjOrFcy00Tg" name="event" direct
          <node xmi:type="uml:ActivityParameterNode" xmi:id="_qcGsN65EecGjOrF
          <node xmi:type="uml:ActivityParameterNode" xmi:id="_DFVgQd66EecGjOrF
          <node xmi:type="uml:ActivityParameterNode" xmi:id="_Ec8zcn66EecGjOrF
          <node xmi:type="uml:CalloperationAction" xmi:id="_HiY6YN66EecGjOrFcy
          <node xmi:type="uml:AcceptEventAction" xmi:id="_Jps1Id66EecGjOrFcy00
            <trigger xmi:id="_MgyeYN66EecGjOrFcy00Tg" name="EVENT" event="_Mgy
          </node>
          <node xmi:type="uml:FlowFinalNode" xmi:id="_Q_FeUN66EecGjOrFcy00Tg"
          <node xmi:type="uml:ForkNode" xmi:id="_v74IcN66EecGjOrFcy00Tg" name=
          <node xmi:type="uml:MergeNode" xmi:id="_XIBygn66EecGjOrFcy00Tg" name
          <edge xmi:type="uml:ControlFlow" xmi:id="_QHMOEN66EecGjOrFcy00Tg" na
          <edge xmi:type="uml:ControlFlow" xmi:id="_R4ADUN66EecGjOrFcy00Tg" na
          <edge xmi:type="uml:ControlFlow" xmi:id="_Y0D0cN66EecGjOrFcy00Tg" na

```

Figure 5.4: XML representation of an ARCTIS building block for the construction of the template for a UPnP ARCTIS building block.

An ARCTIS building block is a model as it is an instantiation of the ARCTIS modeling tool. Similarly, considering that XML is a modeling language then a UPnP service description can be considered as a model since the UPnP description is an instance of the XML language. Both models are in XML.

To transform the XML UPnP service description into an XMI building block, the service presenter uses an XML Parser. The XML parser will parse the properties of a UPnP service (device name, device properties, actions, events, arguments, argument types, and direction of actions). Based on these properties and the XMI template, an ARCTIS building block is constructed. The process is done in three main steps as follows:

1. read the description.xml
2. read the UPnP properties
3. map the properties onto the template of ACRTIS building block.

The algorithm of the service presenter is shown in listing 5.1. The transformation of UPnP service descriptions into ARCTIS building blocks is a kind of text-to-model transformation. For this, the transformation rules presented in Table 4.1 is used. The rules are very simple. For example, to present the name of the building block, we use the name of the UPnP device. Figure 5.5 shows a snapshot of UPnP

Light service in an ARCTIS building block presented in Figure 4.7. The snapshot shows some of the UPnP properties have been mapped into the XMI of an ARCTIS building block. A complete version of the XMI can be found in Appendix C.

Listing 5.1: Algorithm for the transformation of UPnP service description into an ARCTIS building block.

```
read description.xml //UPnP service description

get UPnP device_name
create building_block (device_name)

get device_properties //device url, device number, device friendlyName, ect
for each of property_name
    create new xmi:id for variable
    name=property_name

get actions
    for each action_name
    {
        {
            create new xmi:id for operation
            name= action_name
            get arguments
            for each argument
            {
                create new xmi:id for argument
                name=argument_name
                get type
                create xmi:id for type
            }
        }
    }
}
```

In addition to the ARCTIS building block, the service presenter generates also a Java class. The implementation class is basically used as a proxy for service invocations. To generate this implementation class, a template is used. We use Cyberlink for Java for the UPnP services [52], to construct this template. This implementation class is connected to the ARCTIS building block.



Figure 5.5: XML representation of the UPnP Light service in an ARCTIS building block presented in Figure 4.7. The XML contains information about operation names, xmi:id, arguments, type of arguments, directions, etc.

Listing 5.2: Algorithm for the generation of code connected to the building block. This algorithm is for UPnP services.

```

read description.xml //UPnP service description
get UPnP device_name
create class (device_name)
get serviceType
for each serviceType
{
get actions
for each action
{
create operation(action)
{
get arguments
for each argument
{
get argument_name
get typeOfArgument
get direction
create argument (argument_name,type , direction)
}
}
}
}
}

```

Listing 5.3: The UPnP code for service invocations.

```

public class UPnPLight extends ControlPoint implements NotifyListener ,
    \index{Event}EventListener , SearchResponseListener {

    private final static String
    LIGHT_DEVICE_TYPE = "urn:schemas-upnp-org:device:BinaryLight:1";
    private final static String
    LIGHT_SERVICE_TYPE = "urn:schemas-upnp-org:service:SwitchPower:1";

    public void Power(String deviceType , String i){
        Device dev = getDevice(deviceType);
        org.cybergarage.upnp.Service ser=dev.getService(LIGHT_SERVICE_TYPE);
        newPowerState=i;
        Action setPowerAct = dev.getAction("SetTarget");
        setPowerAct.setArgumentValue("newTargetValue", newPowerState);
        setPowerAct.postControlAction();
    }

    public boolean GetLoadLevelStatus (){
        //code
    }

    public boolean GetMinLevel (){
        //code
    }

    public boolean SetLoadLevelTarget (){
        //code
    }

    public void SetTarget(String newTargetValue){
        Power(LIGHT_DEVICE_TYPE,n);
    }

    public boolean GetStatus (){
        Action getPowerAct = dev.getAction("GetStatus");
        if (getPowerAct.postControlAction() == false)
            return;
        }
    }
}

```

Similar to the generation process of the building block, to generate the implementation class we also have developed a parser. The parser reads the UPnP properties and maps it to the class template. Listing 5.2 shows the algorithm for the service presenter to generate an implementation class connected to the ARCTIS building block. Listing 5.3 is the generated Java class for the UPnP Light service. A complete source code is presented in Appendix D.

5.3 The Service Library

The service presentation in PMG-pro is only for visually presenting run-time services in order to service designers and integrator can use them specifying service-based applications/composite services at design-time. The (visually) presented service models do not describe the internal behavior of the actual services. The actual services and its behaviors that reside in the service providers are hidden and invisible to the developers and integrators, but they have a complete functionality description of the run-time services. Access functionalities to the actual services at run-time are done by the connected code that is connected logically with the visualized service models. This is how the separation of a problem with the access functions to particular run-time environment is realized.

The service library was implemented in a simple table that contains information about the relation between models and its real code. At the moment we implement the service library manually and store as .txt file.

The table contains four columns: the service graphical representation, the level in the service hierarchy, the file name of the connected code and it subclass. Table 5.1 shows the table.

Table 5.1: A simple database for implementing the Service Library for the purpose proof-of-concept.

Service Name	Level	Source code file name	Sub class
UPnP_Lamp	0	UPnP_Lamp.java	-
UPnP_Light	0	UPnP_Light.java	-
DPWS_Lamp	0	DPWS_Lamp.java	-
LampService	1	Lamp.java	UPnP_Lamp, DPWS_Lamp, UPnP_light
LightService	2	Light.java	LampService
UPnPMedia-Renderer	0	UPnPMedia-Renderer.java	-
DPWSMedia-Renderer	0	DPWSMedia-Renderer.java	-
VirtualMedia-Renderer	0	VirtualMedia-Renderer.java	-
MediaRende-rerService	1	MediaRen-derer.java	UPnPMediaRenderer, DPWSMediaRenderer, VirtualMediaRenderer

5.4 Code Generator

Although some existing modeling tools provide built-in code generators, for the purpose of proof-of-concept, PMG-pro also provides a simple code generator. This was done since the built-in code generators, for example the ARCTIS code generator, do not suit for the proof-of-concept. ARCTIS does not implement the providing step of PMG-pro. To provide a service-based application as a new service, an additional code is required.

We have implemented a code generator that is able to generate code automatically from models of a service-based application. For this we adapt the concept that has been presented in Section 4.5. To provide as a UPnP service, the code includes a service monitor that will monitor the availability of services and make an adaptation. At the moment we implemented only for UPnP services. The reason is that the UPnP services have implemented a notification mechanism when a device with embedded services come and go to the network.

To generate code the concept of model-to-text transformation is used. We did not use any transformation language to generate code, but a Java program to transform models into texts (i.e., source code). At the moment, the program can only read collaboration activity diagrams (i.e., ARCTIS diagrams) and sequence diagrams (i.e., Rational Rose). However this is not a problem since most existing modeling tools support these two types of diagrams. We will discuss this limitation in Section 6.1.

Models of a service-based application have two parts; the structure and behavior. The code from the behavior part is generated using the instantiation concept. One instantiation is created for each used service models. In the case of ARCTIS to Java code generation, from each building block, one object is instantiated [55]. Since the building blocks in this scenario are platform independent, the objects to be instantiated are depending on the platform selection.

The behavior part is generated from the interactions between the service models. Information for code generation of the behavior parts is taken from the activity nodes. For this we adapt the generation method presented in [10]. With regard to their method, an ARCTIS building block can be considered as an entity that executes an external action.

Example of the generated code from the structure and behavior parts is presented in the case study 3, where we compose different services and provide the functionality of the service-based application as new UPnP service.

5.5 Case Studies

For the purpose proof-of-concept, we use a smart home environment. Smart homes are a simple example of environment containing different devices (with embedded services) that can be freely controlled by others, in the sense of service invocations. Using this feature, new application can be developed by defining a set of service interactions either using event-based or request/reply interaction styles or both. It is also considered that there are several services that can come and go in the smart house environment.

A smart home environment was also used in [109] and [22]. Figure 5.6 illustrates the smart home that is used in this thesis. A residential gateway is controlling and managing home devices with embedded services. In this type of dwelling, it is possible to maintain control of doors and window shutters, valves, security and surveillance systems, etc. It also includes the control of multi-media devices that are parts of home entertainment systems.

Three similar cases studies are presented in this section. For each of case studies, a composition scenario has been developed. The aims of the case study is showing how the PMG-pro method can be applied for different domains and to show that different modeling languages can be used. The three case studies are:

1. Model-driven approaches for the development of smart home services

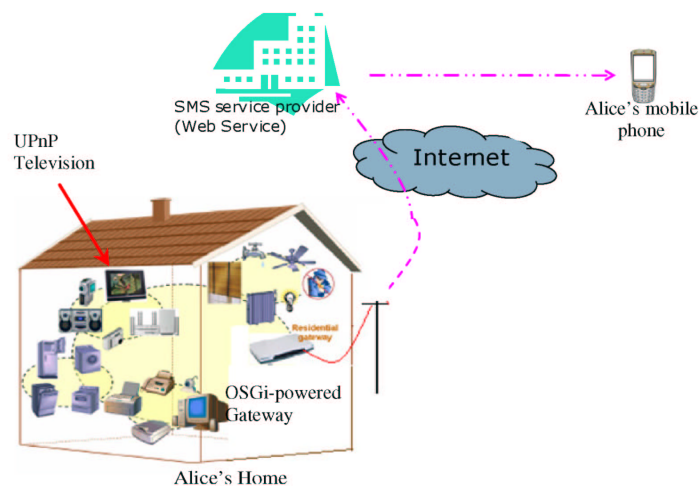


Figure 5.6: Smart homes - Selected environment for the purpose of proof-of-concept. All devices provide services that can be invoked. We have three case studies based on this environment. Each case study has a scenario of service-based application.

2. The development of event-based applications using existing services
3. A model-driven method for the development of service-based applications in a heterogeneous services environment

In these case studies, we use mainly UPnP services as a model of software unit. The reason is that UPnP uses XML to describe a service and, it seem become a common to use XML to describe services (e.g. web services). Moreover, UPnP has been widely used and implemented in several home devices.

5.5.1 Case Study 1: Model-driven Approaches for the Development of Smart home Services

This case study presents our idea of applying MDD for the composition of existing services, by which we aim at demonstrating how new smart home services will be promoted. Similar case study was published in the Proceeding of *Ambient Assistive Health and Wellness Management in the Heart of the City, ICOST 2009* [94]

5.5.1.1 The aim of the Case Study

The aim of this case study is to demonstrate the use of the MDD approach for the developing a smart home service by composing different services that have been implemented using different technologies. For this purpose, we use Rational Rose as a tool. Therefore, existing services are abstracted into UML classes. Then, a sequence diagram is used to model the new service-based application.

5.5.1.2 Introduction

Smart home is about the application of automation techniques for the comfort and security of residents' privately owned homes. In smart homes, a residential gateway (RGw) connects the home with different embedded devices (and services) to the Internet. We consider all those services to be basic services in which we can promote a new advanced service by means of service collaboration of those basic services. The new service may include multiple devices, both internal services in the home network and external services outside the home network.

5.5.1.3 A Use Case Scenario

In this scenario, we assume that there exist four different services running independently in a smart home. Among of them are a UPnP AlarmModule that pro-

vides data services, a UPnP_mobile_phone detection service that provide a mobile-absence detection service, UPnP_based television that provides display service and a SMS_Web service. We will develop a new application that enables the owner of a smart home to get the status of her/his health-related home equipment (represented by sensor in the alarm system) wherever she/he is. An alarm message will be displayed onto either the UPnP_based television when the owner is at home or on his/her mobile phone when the owner of the house is not at home. The new service is implemented as an OSGi service.

```

<?xml:version="1.0" encoding="utf-8"?>
<s:Envelope s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
xmlns:s="http://schemas.xmlsoap.org/soap/envelope">
  <s:Body>
    <u:Display xmlns:u="urn:schemas-upnp-org:service:TVDisplay:1">
      <Text>Sensor in the kitchen in ON</Text>
      <start_X>20</start_X>
      <start_Y>100</start_Y>
    </u:Display>
  </s:Body>
</s:Envelope>HTTP/1.1 500 Internal
Content-Length: 604
    
```

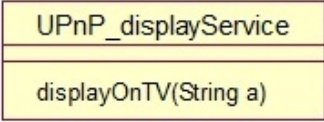


Figure 5.7: An abstract representation of the UPnP display service in an UML class.

5.5.1.4 The Development Process

The first step is service abstraction in terms of UML classes. There exist several frameworks of service description, for example, WSDL(Web service description language), UPnP, and OSGi. Based on a service description, we present a service as a UML class and generate code to implement the service invocations.

The transformation process of a UPnP service description into an UML class is done by applying the concept of model transformation. For this transformation, the transformation rules presented in Table 5.2 are used.

Table 5.2: Transformation rules between UPnP - UML class

UPnP	UML Class
Device name	Class Name
Action	Operation
Action argument	Operation’s argument
State variable	Attribute
Type of state variable	Parameter type

Using the table above and a template of UML class (in XMI), the `displayOnTV` action is transformed into `displayOnTV` operation. The action arguments `x`, `y` and `message`, are transformed into operation arguments. Table 5.3 shows the transformation result. Figure 5.7 shows the result of the transformation when the XMI of the UPnP-based Television description is opened in Rational Rose. Similar abstraction process is done for the other services.

Table 5.3: A transformation of an UPnP service description into an UML class

UPnP	UML Class
Device name: UPnP_displayService	Class Name : UPnP_displayService
Action: displayOnTV	Operation: displayOnTV
Action argument: 1. X: int 2. Y: int 3. a: String	Operation's argument: 1. X: int 2. Y: int 3. a: String

After all the existing services we want to include in the new service have been abstracted into UML classes, we can model the new service, both of the structural and behavior aspects. Thus, we have 4 UML classes representing four existing services. Figure 5.8 below shows a sequence diagram of the new service. The new service, `alarmService`, is a collaboration of four different services (the display services, the SMS Services, phone detection service and the alarm service).

The behavior of the `alarmsystem:OSGiService` is described in a simple sequence diagram. The main purpose of a sequence diagram is to define event sequences that result in some desired outcome. The focus is less on messages themselves and more on the order in which messages occur; nevertheless, most sequence diagrams will communicate what messages are sent between a system's objects as well as the order in which they occur. The dashed line arrows indicate that the messages are type of event, which are in this case, UPnP events. A solid arrow is used to describe that an object calls an operation on the other object.

From the figure, it can be seen that when a sensor is activated, the `alarmsystem:OSGiService` will search the location of the sensor and detect the presence of the mobile phone. The application will then sendSMS and display the message to the display service on the UPnP television. The use of dashed arrows to describe events may help the implementation of the models, but it would become very complex using sequence diagrams to describe complex event-based systems. An example of other difficulty of the use of sequence diagram is the describing decision. For

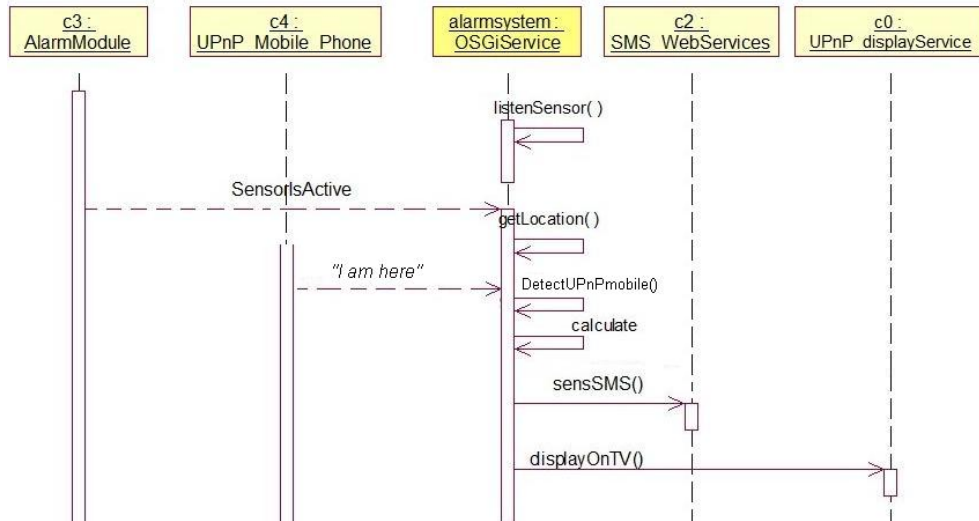


Figure 5.8: An UML sequence diagram of the alarm system in the scenario.

example, to describe a case when the mobile phone is at home the message will be sent onto the UPnP-based television. Otherwise, it uses the sendSMS service to reach the mobile phone.

To execute the models, the transformation of the model to code is done. For this, a template-based source code generator was built using Java. Since the combined services will be provided as an OSGi service, the template was developed based on OSGi framework, which is in this case, we use the Knopflerfish OSGi framework.

5.5.1.5 Summary of the Case Study

In this case study, we have demonstrated the MDD approach for developing a smart home service by composing four different services, in different computing devices; UPnP services, OSGi services, and Web Services. The new service is an OSGi service which is intended to run on an OSGi-powered residential gateway. The problem with the use of UML sequence diagram is its limitation to express complex event-based system.

5.5.2 Case Study 2: The Development of Event-based applications using existing Services

We have made a case study of developing an event-based application from services specified using UPnP specifications. In this case, a service-based application is developed. The combined functionality of the new service based application is not

provided as a service. It is just a normal application that is run and demonstrated on a single computer. The case study was published in the Proceeding of *2nd International Conference on Computer Technology and Development (ICCTD 2010)* [92]

5.5.2.1 The aim of the Case Study

Service interactions in service-based applications might use either event-based or request/reply-based interactions or both. We are interested in event-based interactions. Furthermore, we are interested on the use of model-driven approaches for the development of event-based applications. The aim of the case study is to show that another tool can be used to develop service-based application. Instead of Rational Rose, we use ARCTIS as modeling tool.

5.5.2.2 Introduction

Event-based systems are widely used for the development of reactive systems. The development of such systems is often be done by integrating compound, autonomous, loosely coupled software units (components), including sensors, device controllers, and databases.

In this case study we use UPnP services as an example of XML-based service description. We consider that UPnP devices in a smart home provide services can be used by other devices (a control point). Thus, a control point is a service user. From the perspective of a service user, an UPnP device can be presented as an object. They have attributes (i.e. state variables) and provide functionality/services/operations (i.e. actions) to the service users. From the perspectives of software developers they can be presented as objects that can be composed with other objects. In PMG-pro, the presentation of UPnP devices as an object is done during the presentation step.

As a normal service user, a control point can use one or more UPnP services. In the object-oriented perspective, the connections between an instance of a control point (an object) and instances of UPnP devices (objects) can be modeled differently, for example using an object diagram. This is how we consider that UPnP is compatible with object oriented environment.

This section gives a background for the UPnP technology, in particular, the idea of event-based systems in UPnP-based smart homes.

1. A Universal Plug and Play (UPnP)

An important aspect of smart home networks is self-configuration in which

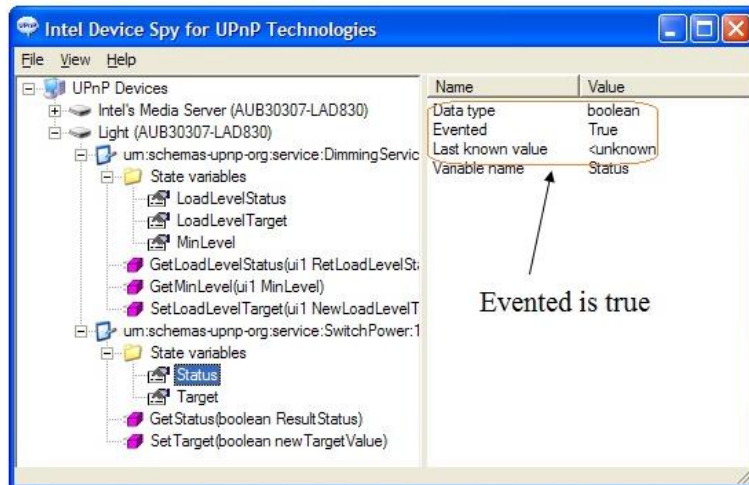


Figure 5.9: Events of the UPnP Light service on Intel's SDK for UPnP-Device Spy.

autonomous devices are allowed to join and leave the home network and at the same time, the network automatically learns about any changes. UPnP is one example of the enabler to achieve this purpose. UPnP allows flexible connectivity to ad hoc and unmanaged networks.

- UPnP Services

It is a requirement that every UPnP-based device must host an XML device description that lists specific properties about the device and the services associated with the device. For example, it contains information about the FriendlyName of the devices, the ID of the device, etc. The device description document also includes a Uniform Resource Locator (URL) for the service description. The service description is an XML document that lists the actions and state that apply to a specific service offered by the device. See Appendix B for a complete example of UPnP service and device description presented using XML document.

- UPnP Events

One thing that is interesting with the UPnP is that UPnP services have state variables that can be used for building event-based systems. When a state variable has changed its value, it will send the changed value to all other UPnP devices to tell that the value has changed. Figure 5.9 below shows event description in Light UPnP device.

A UPnP device might act as a control point or a device. UPnP devices that act as a control point can register for events to receive notifications if the state variables for a particular service description changes. The publication of such states changes by the service is called eventing. In this context, the evented variables are those variables that send a notification to a control point regarding such a change of status.

A control point can subscribe to a number of evented variables. When subscribed to a particular event, the device submits the complete state of all variables to the control point. After sending the complete set of information about all variables to the control point, the device keeps sending updates to the control point if any changes in the specific variables occur.

2. (UPnP-Based) Event-based Systems

UPnP service description contains a description of state variables (in object-oriented term is attribute). These state variables can be type of evented, which means that, any changes to the value of these variables will be confirmed to all control points (service users) that subscribe to it. With these evented state variables, a control point (a service user) can execute necessary operation that depends on that event. This is how we consider that UPnP services can be used to build a reactive system. That is a system that depends on event to execute operations/functions.

A composite service is a service-based application that composes services and provides new functionalities to others as a new service. In our case, we consider all UPnP services to be basic services that we can compose as new service-based applications. One simple example of a composed application in smart homes is a weather-to-music application that will play a specific song or music for a specific weather condition. In this example, an UPnP device that measures the weather parameters will give the measured data to a UPnP media renderer to play a specific song.

A simple event-based system consists of a set of producers and consumers which are clients of an event notification service. All clients are connected to the same service and access its functionality in order to subscribe to or publish notifications. As mentioned earlier, UPnP has all mechanism to publish, subscribe, and unsubscribe that enables for building event-based systems.

3. Extending UPnP Scope Using Event Delivery Gateway.

To distribute the UPnP events, an event delivery gateway is required. We have implemented UPnP event delivery gateways (u-EDG) using Java. Figure 5.10

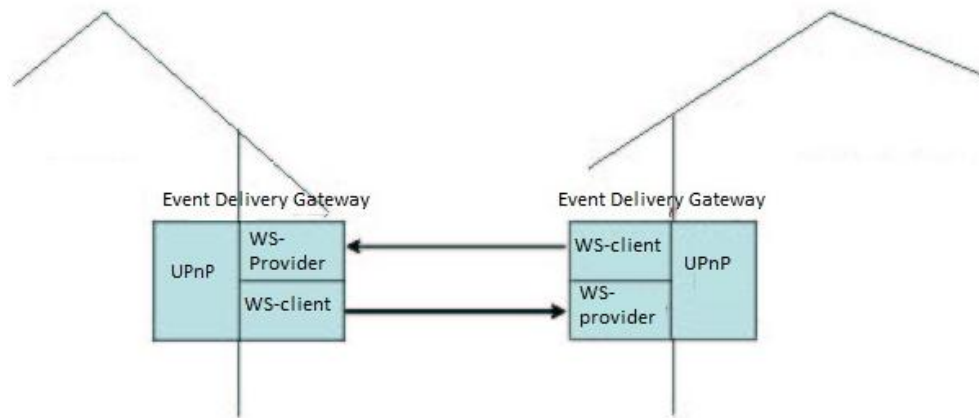


Figure 5.10: An extending scenario of the UPnP network.

shows the connection of two smart homes using event delivery gateways. The u-EDG contains of a Web service provider, a Web service client, and an UPnP EventListener. The Web service provider provides a service that can be used by a client to upload UPnP events and UPnP packets (a SSDP packet). For the definition of UPnP events and UPnP SSDP packet the reader should refer to [46].

On the client side, a Web service client can use the provided services to upload received packets and notifications from one smart home to another. A UPnP EventListener which is on the client side listens to UPnP events and UPnP packets. When an event or a UPnP packet in a smart home has been received the Web Service client will send them to other smart homes using the provided Web service. To be able to capture UPnP events, the client must also implement UPnP control point [46].

Every UPnP event has device ID (uidd), sequence number of the event (seqd), name of the state variable (named) and the value (valued). With the event delivery gateways, UPnP events can be distributed from one smart home to other smart homes. With this distributed mechanism, the numbers of devices that can be composed is also increasing.

5.5.2.3 A Use Case Scenario

We assume that there exist different services running independently in a smart home. Among of them are a UPnP WeatherModule service, a UPnP light service,

services from a UPnP media renderer and services from a UPnP media server. We want to develop an event-based application that will play mp3 music/song on selected track number when the room temperature is greater than 20 degree and the light (a lamp) is turned on. Here, the lamp status is used to indicate that the owner of the smart home is at home.

To give an explanation of the services functionalities, we present a short description of each of the service as follows.

1. The UPnP WeatherModule Service

This service has four outputs; air temperature, humidity, solar radiation, and wind speed. Any control point that subscribe to this service will get notifications when the value of the outputs are changed. This module is an event producer.

2. The UPnP light service

The UPnP light service has five inputs (actions) and two outputs. The actions are for example; SetLevel and GetLevel. Despite act as an event consumer, this UPnP light can also act as an event producer. The state variables Status and LoadLevelStatus are an event. For a detail description of the UPnP light service the reader should see [46].

3. The UPnP Media Server and Media Renderer

Any media in a media player should be able to be played in media renderers. In this case study we assume that the media renderer has already a list of media, so the control point needs only to send commands (control) to play, stop, previous, and next. For detailed example of how a UPnP media server and a media renderer are used the reader should refer to [34, 33].

A media server and a renderer might act as event consumer and producer, but in this case study we use them as an event consumer. We assume that the media renderer has already a list of song, so a control point can send a command control to play, stop, next, etc.

5.5.2.4 The Development Approaches

To develop event-based applications using UPnP services, it is necessary to have a model of the UPnP services.

1. Service Presentation/Abstractions

Different presentation can be use to present UPnP service as an object. To

represent a service by models we need a modeling language. Instead of using a UML class, we use an ARCTIS building block. A building block is a kind of encapsulation of activities that has external inputs outputs.

Table 5.4: A Transformation of an UPnP service description into an ARCTIS building block

UPnP	Building Block
Device name: UPnPLight	Building block name: UPnPLight
Action and argument: 1. GetStatus(boolean ResultStatus) 2. SetTarget (boolean newTargetValue) 3. GetMinLevel(int MinLevel) 4. SetLoadLevelTarget(int newLoadLevel-Target) 5. GetLoadLevelStatus(boolean retLoad-LevelStatus)	Parameter and its type: 1.a getStatus:String* 1.b resultStatus:boolean ** 2. setTarget:boolean* 3.a getMinLevel:int* 3.b MinLevel:int 4. setLoadLevelTarget:int* 5.a getLoadLevelStatus:boolean* 5.b retLoadLevelStatus:int**
State variable and its type: 1. Status:boolean 2. LoadLevelStatus:int	Parameter output event with type: 1. status:boolean ** 2. loadLevelStatus:int**

**Parameter type of Output event. *Parameter type of Input event

Services abstraction is a kind of text-to-model transformation. By using service descriptions, existing and running services in the network can be transformed (abstracted) using graphical presentations. In addition to graphical representation, a Java class is generated. This class is used as a proxy to make invocations to the services that resides in a service provider.

The actual process of model transformation is done by applying text-to-text transformation. That is XML-UPnP service description into XMI-ARCTIS building block. We have defined a template for the XMI-ARCTIS building block as presented shortly in Figure 5.3. Using this template and by applying the transformation rules presented in Table 4.1, the XML-UPnP service description can be transformed into XMI-ARCTIS building block. The XMI then can be opened in ARCTIS modeling tool. Table 5.4 shows the transformation process of UPnP parameters into ARCTIS building block parameters. A complete XMI file and its representation in ARCTIS of the UPnPLight is shown in Appendix C.

2. The Model of Event-based application

After all existing services mentioned above are presented in building blocks

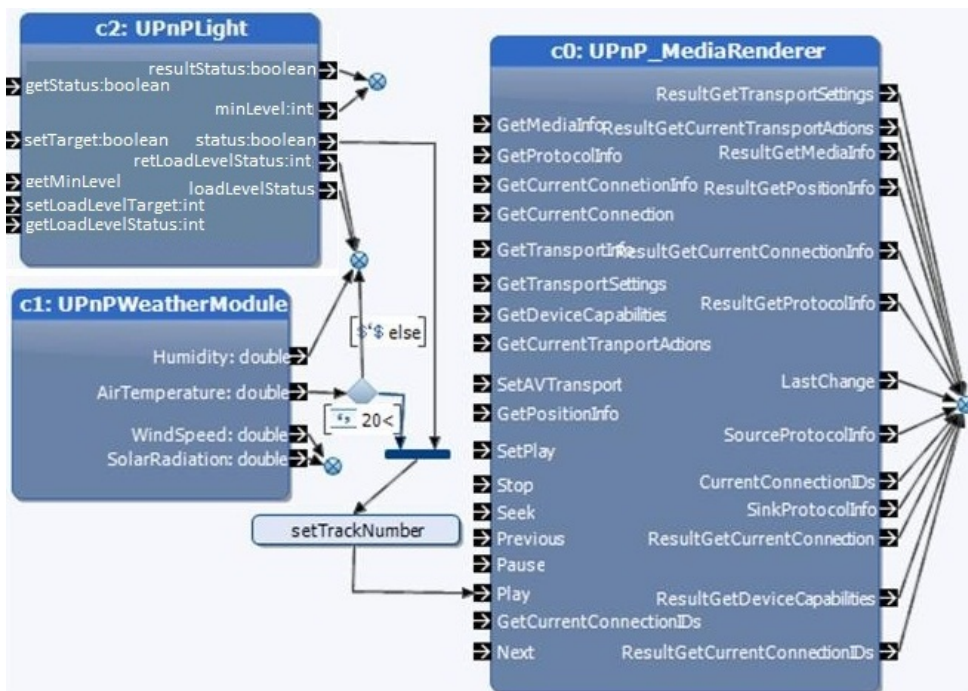


Figure 5.11: The feelgood model is presented in an ARCTIS building block diagram. Three building blocks are used to describe the combined application.

by the service presenter (abstractor), the modeling of the new service-based application can be done. With building blocks, a system is defined in terms of building blocks composition, where the building blocks themselves encapsulate activities. Accordingly, a model of UPnP-based event-based application is specified by defining connections of UPnP building blocks. The model defines the structure and behavior of the event-based applications.

Figure 5.11 shows a conceptual model of service-based application presented in building block diagram. The figure shows that a service based application has one or more building blocks that are connected using connectors. The connections between building blocks represent service interactions. They are defined using activity nodes in a UML activity diagram. As shown in Figure 5.11, an ARCTIS model of the new event-based application is defined. If we look at the building block C2:UPnPLight, the **Status** state variable of the lamp will be used to send a play command to the C0:UPnP_MediaRenderer if the air temperature is greater than 20 degree.

3. Code Generation.

From the model that is shown in Figure 5.11, our concern is how the code

for implementing the event-based application can be generated out from the connections of building blocks automatically. A connection may mean an invocation of one device to another service that is embedded in a device. For this purpose, a code generator can use a simple way of code generation by interpreting directly the activity diagram. Since we have all necessary code for each building block, the generated code acts as 'glue code' that coordinate the included services (devices).

ARCTIS has a code generator for J2SE that enables automatic code generation from ARCTIS model [55]. In our prototype, one Java class will be generated for one ARCTIS building block diagram. The Java code for an ARCTIS model is generated based on the ARCTIS models, templates, and proxy classes. The class acts as a 'glue code' that implements the service composition.

5.5.2.5 Summary of the Case Study

In the case study we have demonstrated how the model-driven approach is used for the development of such systems. We abstract and present the services into reusable ARCTIS building blocks that we can use to build and specify a new event-based application. A building block is a graphical representation of implementation class that is reusable and support for incremental development.

With model-based development approaches, the specification of service interactions can be done at a high abstraction level. Although in this use case we use only UPnP and Web services, we believe that the approach is also applicable for other abstract services that are XML-based. Of course, in this case, objects in the Internet of Things should provide embedded services that are described in XML.

5.5.3 Case Study 3: A model-driven method for the development of service-based applications in a heterogeneous services environment

In this case study, we developed a service-based application using different service technologies. Moreover, the service-based application is then provided as a UPnP service. This case study is the most complete of service creation cycle of the PMG-pro method. In this case study, the combined functionality is provided as a UPnP service. Thus, the service is described using XML, which is more flexible than OSGi services in terms of interoperability. The case study was published in the Proceeding of *IEEE International Conference on Software Engineering and Service*

Sciences ICSESS 2010 [97]. A full paper of this case study was also published in SDL forum.

5.5.3.1 The aim of the Case Study

The aim of the case study is showing how the PMG-pro can be used to model service-based application in a heterogeneous service environment.

5.5.3.2 Introduction

The fact that the concept of service-orientation systems can be defined, implemented, and described in several ways has introduced different types of implementation of the concept. For this and other reasons, the motivation of this work was to provide a method for building service-based applications using services that were developed using different technologies (a heterogeneous service environment).

5.5.3.3 A Use Case Scenario

Considering that all the devices in the Internet implement their functionalities as embedded services that can be invoked, new composite applications can be promoted. In this scenario, we use the following devices:

1. WeatherModules that provide different data collection services (i.e., air temperature, solar radiation, wind speed, and humidity sensors).
2. Lamps that provide on-off and dimmer services.
3. Media Renderers that provide playing of multimedia services.
4. Virtual devices that provide sending e-mail services.

In a smart home environment, different open and standardized languages and technologies are often used to describe services. For example, Universal Plug and Play (UPnP) is one of the popular standards for describing embedded services of home entertainment devices (e.g., media renderer). Other examples are Web Services Description Language (WSDL) and Device Profile for Web Services (DPWS). Based on these different embedded services, new applications can be promoted. In this scenario, a new application is promoted. The application has new functionalities as follows:

- the application is able to send notifications (e-mail) when the air temperature from the WeatherModule is greater than 50°C,

- it will play music/songs on the media renderer when the light is turned on and the solar radiation is below than 10, and
- provides two new UPnP services. The first service is to enable a user to configure on which song user want to play for a specific weather condition, while the second service is to get the configuration info.

5.5.3.4 The Development Process

1. Presentation.

The presenting step is a kind of text-to-model transformation. We have developed a service abstractor/presenter that is able to transform UPnP service descriptions into ARCTIS building block and its bound source code (Java class). ARCTIS is a modeling editor that uses a building block as a basic entity.

A UPnP device has two kinds of description; device and service description. A UPnP device can have several services that in a UPnP service description is called Actions. To automate this step we use transformation rules. Table 4.1 in Section upnptoARCTIS shows transformation rules to transform different properties in a UPnP service description into properties in an ARCTIS building block. To construct the transformation rules, both ARCTIS and UPnP meta-models are required. However, the rules are very simple. For example, to present the name of the building block, we use the name of the UPnP device. Obviously, other XML-based service descriptions (e.g., WSDL, DPWS) will use a similar process.

In addition to the graphical representation, source code (a Java class for service invocations) is also generated for each UPnP service description. Obviously, other programming language can also be used. After all service descriptions are transformed into graphical service models the modeling step can be started.

2. Modeling

Using the service taxonomy it would be possible for the modelers to model new service-based applications using service models at any level of hierarchy. The higher the level of hierarchy the more platform-independent the models would be. By high-level models, we mean models of service-based applications that are built using platform-independent service models at level 1 and above of the hierarchy in the service taxonomy. From high-level models of service-based applications different source code can be generated. Obviously,

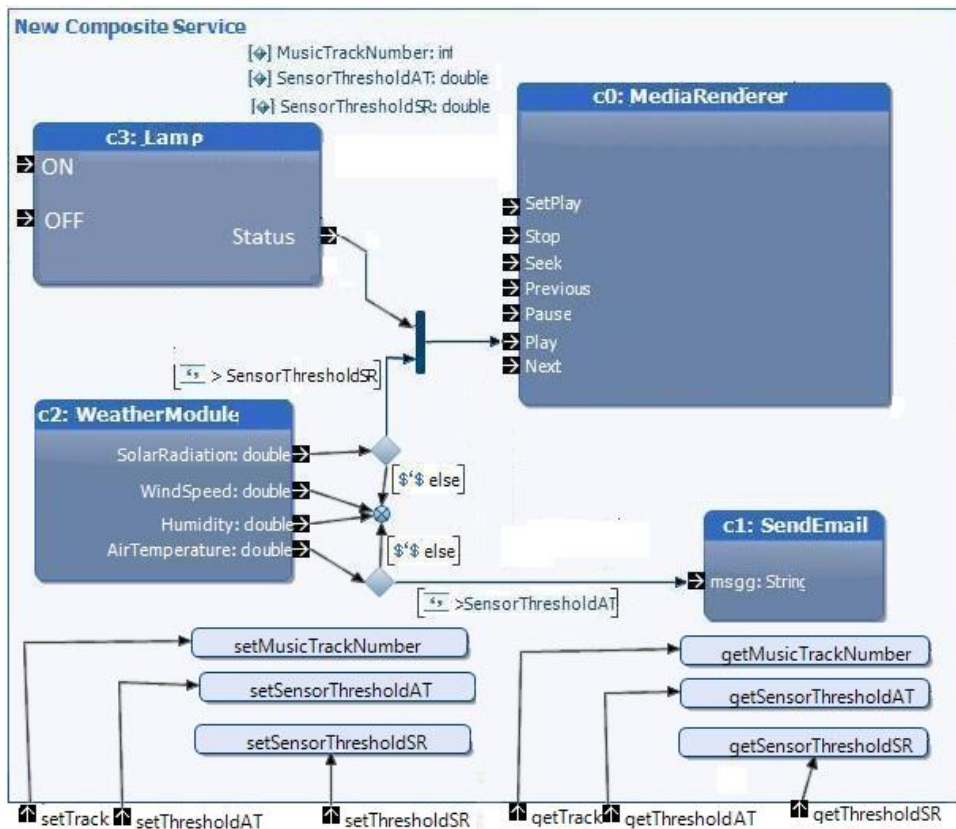


Figure 5.12: The ARCTIS model of the new application. The included service models specify the structure of the application, while the interactions between service models specify the behavior.

it will be limited by the number of source code binds to the service models that are stored in the service library.

Using ARCTIS, a service-based application can be specified as a building block diagram. The behavior of the new service-based application is defined by interactions between building blocks, which are in this case defined using activity nodes defined in UML 2.0. Accordingly, the semantic follows the semantic of UML 2.0 activity diagrams. Figure 5.12 shows an ARCTIS model of the service-based application defined in the scenario. There are four building blocks that represent different existing services mentioned in the scenario. In this model, the service models are taken from the level 1 of the service hierarchy. This means that they may have several different implementations.

In ARCTIS, a service-based application can be, again, encapsulated and presented as a new building block. This allows us to model the new provided

services defined in the scenario. As defined in the scenario the new service-based application will provide two new services (i.e., SettingService and Get-Configuration Service). In ARCTIS, this can be specified by defining new parameters (i.e., ports). Obviously, this new building block can be used to build a new building block diagram as new specification of a software system. This is the way how an incremental development of a large application can be done in ARCTIS.

Listing 5.4: The structure part of the New Composite Service

```

public class New_Composite_Service {

private UPnP_MediaRenderer c0;
private SendMail c1;
private UPnPWeatherModule c2;
private UPnP_Lamp c3;
MusicTrackNumber int;
SensorTresholdAT double;
SensorTresholdSR double;

public New_Composite_Service() {
// main behavior
}

public void setMusicTrackNumber (int number){
this .MusicTrackNumber=number;
}

public void setSensorTresholdAT(double sensorTresholdAT){
this .SensorTresholdAT=sensorTresholdAT;
}

public void setSensorTresholdSR (double sensorTresholdSR){
this .SensorTresholdSR=sensorTresholdSR;
}
public int getMusicTrackNumber (){
return this .MusicTrackNumber;
}
.....

public static void main(String [] orgs){
new New_Composite_Service ();
}
}

```

3. Code Generation and providing

To illustrate the code generation in PMG-pro, we use Java programming

language. To generate code from the structure the block diagram and building blocks are used. From a building block diagram the main application (class) is generated. The name of the class is defined using the name of the building block diagram.

From each building block, one object is instantiated. Since the building blocks in this scenario are platform independent, which objects instantiated are depending on the platform selection. Listing 5.4 shows an example of code when the UPnP platform is selected for the code generation at generating step. Only classes that implement UPnP services are instantiated.

Code from the behavior part is taken from the activity nodes. For this we adapt the generation method presented in [10]. With regard to their method, an ARCTIS building block can be considered as an entity that executes an external action. For example, for the decision node (with air temperature input) produces the code shown in Listing 5.5.

Listing 5.5: An example of code for the behavior part

```
if (c2.airtemperature >= SensorTresholdAT) {
    c2.msgg=SetMessage();
}
else
    break;
```

For the scenario example, two new services are provided as UPnP services. For this, UPnP code must be added. We have implemented a code generator to generate code for the application scenario. Figure 5.13, is a screenshot of the running UPnP services. We use deviceSpy software provided by Intel Tool for UPnP technology [46]. It can be seen that the new application (composite service) provides two UPnP services that are the `GetConfiguration()` and the `SettingService()`.

5.5.3.5 Summary of the Case Study

Software applications tend to be more service-based. Being service-based, two things are required: a way to create simple service and a mechanism for describing the interactions of those simple services. Furthermore, with model-driven development approaches we need to present services and their interactions graphically.

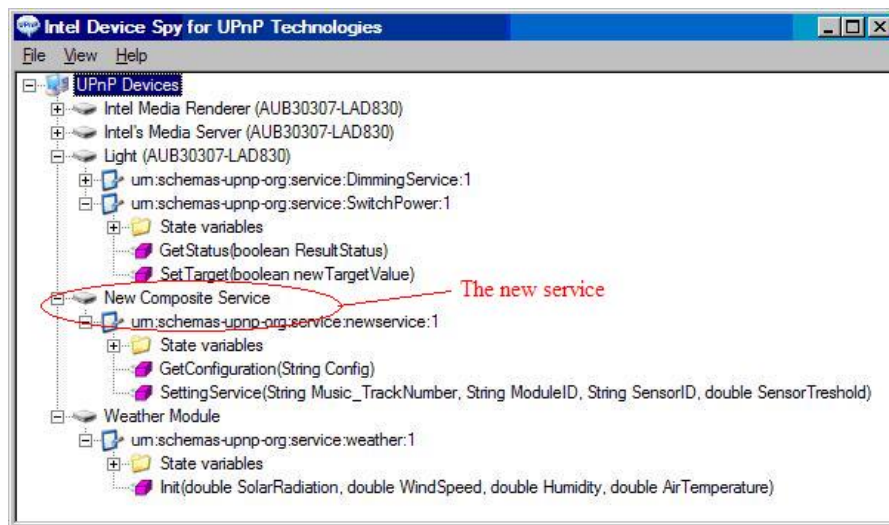


Figure 5.13: The new composite service at run-time. The new service-based application is provided as a new UPnP service.

Several different implementation of the concept of service orientation exist. PMG-pro presents a service as a reusable building block independent of which implementation of the concept. A building block refers to an implementation class that acts as a proxy to an actual concrete service. Interactions between building blocks are defined using activity diagrams. A building block might also include other building blocks. With this approaches, a small, verified, and validated encapsulated activity diagram (a building block) can be used to build incrementally a larger building block diagram presenting a complex software application.

Chapter 6

Evaluation

This chapter is organized in three sections. Section 6.1 presents an evaluation of the PMG-pro method and use case studies. An evaluation of the method against the seven sub research questions is presented in Section 6.2. At the last section, Section 6.3, an evaluation against related work is also presented. For this evaluation we present four related work. The aims of the evaluation against the related work are to show the originality of the method and positioning, that is the contribution of the method in the area of MDD, and software development in the Internet of Things.

6.1 Evaluation of the Method and Case Studies

PMG-pro combines bottom-up, component-oriented and model-driven development approaches to promote a rapid and automatic development of service-based applications. With this combined approach, existing services are presented using graphical notations (i.e., models) conforming to a specific modeling language (the presentation step), then new service-based applications are defined/specified using those models (the modeling step), and finally code for implementing the new software-based applications can be generated automatically (code generation step), see also Figure 6.1.

The approach is intended for service integrators to be able to compose run-time services at design-time, and to promote new services. The main idea is the construction and use of the hierarchical service models in the service library. With this approach, service integrators can focus on the problem solution on a high abstraction level using models, while the implementation will be done by a machine in automated manner.

The PMG-pro method also proposes a new method of handling device capabilities and configurations in the software development for personalized and embedded

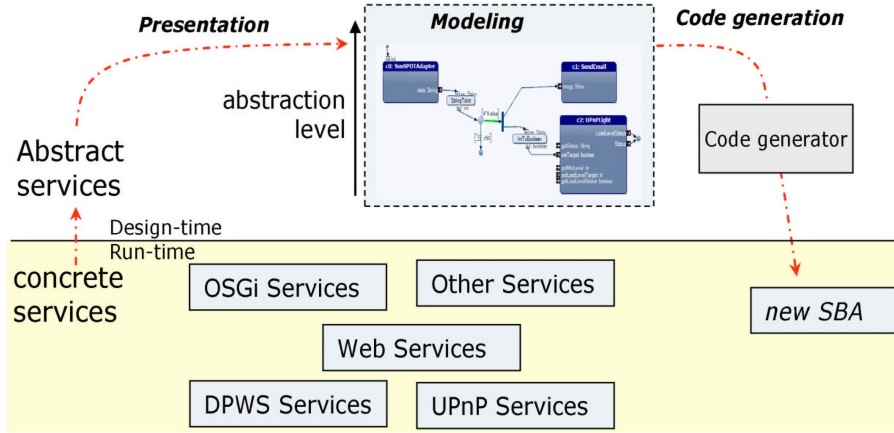


Figure 6.1: The present, model, and generate-provide method for building service-based applications.

devices. The proposed method is different with the existing methods, for example software product lines (SPL). In SPL, software artifacts for a specific device capability and configuration are created only if the use of the software artifacts is predicted in one or more products in a well-defined product line. A library concept is not used in SPL. However, although PMG-pro uses the concept of library, it contains only a limited number of service models that were constructed in a hierarchical manner.

For the code generation from models, two approaches are proposed. The first is to generate code for possible aggregated services in the ontology. In PMG-pro, a service model relates to different implementations of service invocations. These different implementations represent different platforms (e.g., different device capabilities and configurations). All possible device configurations would also support run-time adaptation in case of services are removed or new services appear.

The second possible solution is to generate code only for the present (specific) services. Using the service library, tailored code can be generated for a specific device capability and configuration. This solution requires information about what devices are available at runtime. This means code cannot be generated before the platform information is given, which essentially means code generation on-demand.

For the purpose of proof-of-concept we have developed prototypes of parts of the method. The proof-of-concept has some limitations. For example, currently the PMG-pro code generation only handles activity diagrams and sequence diagrams.

This is not a problem since most of the existing modeling tools support these two types of diagrams. In addition, several modeling tools come with built-in code generators and PMG-pro can use them.

However, since the existing modeling tools are not suitable for the purpose of proof-of-concept where a service-based application is provided as a new service, PMG-pro provides also a code generator. With this additional code generator, service developers have a choice either to use built-in code generators or to use the PMG-pro's code generator. In the first case, PMG-pro facilitates the presentation of existing services into the modeling editor of the tools. This has been shown and demonstrated in case study 1 and 2.

Other example of the limitations of the proof-of-concept is the number of supported tools. The proof-of-concept only supports for ARCTIS and UML Rational Rose 2000. This is not a big problem since other UML-based modeling tools (e.g., SysML and SoaML) use also XMI which is similar to the one used in ARCTIS and UML Rational Rose. Expanding the number of supported tools can be done by developing service presenters in a similar way, but with different XMI target formats and versions.

The capability of transforming only XML-based service descriptions (i.e., UPnP service description) is another limitation of the proof-of-concept. However, this is not a serious problem, since it seems that a service description tends to be a static description and XML tends to be used for such description. By developing different parsers, different service descriptions that are XML-based can be presented visually in a similar way.

Based on the prototypes, three cases study have been performed and presented in previous chapter. In this evaluation chapter, the three case studies are evaluated. The evaluation will focus on the use of the proposed methods.

6.1.1 Case Study 1

In the first case study we have demonstrated the use of UML sequence diagram to model a service-based application, that is an application that is built based on existing services.

Promoting smart home applications using embedded systems is gaining importance, as it can produce composite services with features not present in the individual basic services. Several methods, framework and tools of service composition in a smart home environment have been proposed over the years to address these issues [49, 17, 67, 72]. In [67] for example, they developed a framework to implement the service composition of networked home appliances and entertainment.

However, none of them use MDD. It is found in [5, 9] that shows a method of how the MDD approach is used to compose services, but they focused on web services and did not consider others services technologies that exist in a smart home network.

The result of the case study shows that model-driven development approaches can be used to compose different technologies of service-oriented systems. The composed new service is then being provided as a new service using particular service technologies, which in this case study is OSGi. The requirement is that the run-time services have to be described using well-known standards for describing services.

An important note is that the semantic of UML sequence diagram is very restricted, in particularly, it can be very difficult to model event-based systems. However, the sequence diagram is a good diagram to use to document a system's requirements and to flush out a system's design. The reason the sequence diagram is so useful is because it shows the interaction logic between the objects in the system in the time order that the interactions take place.

6.1.2 Case Study 2

In the second case study we have developed a service-based application that only uses UPnP services. We developed an event-based application by employing the *Event* properties of UPnP. The application is run and demonstrated on a single computer.

Event-based interaction styles have become prevalent for large-scale distributed applications [16] due to the inherent loose coupling of the services. As it is stated in [16], the event-based interaction carries the potential for easy integration of autonomous, heterogeneous software components into complex systems that are easy to evolve and scale. For this and other reasons, a number of event-based middleware infrastructures were developed for example are in [16, 15, 23]. Unfortunately there is only a little literature on the development approaches and methodologies of event-based systems using embedded services. If there exist, most of them are implementation-oriented and do not use MDD.

In this case study, we use ARCTIS modeling editor to model the interactions between services, where collaboration and activity diagrams are used. The activity diagram provides activity nodes (such as actions, pins, data and control flows, and signals) that allow specifying the meaning of a behavioral element (such as the body of an operation from the class diagram, or the effect of a state transition from the state diagram). With these, modelers(i.e., service integrators) are allowed to specify behavior of a new event-based system in a precise way.

The result of the case study shows that an automatic graphical representation of existing (run-time) services at design-time is an important step of applying the concept of reusability and the use of MDD. Depending on the capability of modeling tools, code can be generated automatically from models. However, the code generation can only be automated if each of the generated abstract service models has a relation (directly or indirectly) with code for implementing the service invocations.

6.1.3 Case Study 3

The whole concept of PMG-pro is shown and demonstrated in this case study. In this case study, we have developed a similar application as previous case studies. However, we provide the functionality of the service-based application as a UPnP service for other users. All services are presented, a new service is modeled using existing modeling language, and code that includes additional code for providing the application as a new service is generated automatically.

In this case study we propose to use ontology for the construction of service hierarchy. The basic idea is that for all similar services it will have a common service model. This common model has one higher abstraction level. With this idea, service models can be constructed in hierarchical service models (i.e., at different service abstraction levels). Accordingly, using these service models, service integrators can model service-based applications at different abstraction levels. In the case study, a new service-based application is specified using level 1, where a common service model has several possible sub service models.

In this case study, we also show two ways of using the constructed platform models for the purpose of code generation. Firstly, based on the information of the targeted platform, the code generator uses the platform models to generate code tailored for the selected target platform. Alternatively, the generated code includes code that contains a detection mechanism to do necessary adaptation at run-time. The fully automated code generation is possible, since the service models are connected to code for implementing the service invocations.

The result of this case study shows that providing a service from a service-based application requires the existing service frameworks and APIs. However, more work must be done since the construction of the hierarchical service models was done manually. Moreover, a template is also required for each different service technology.

6.2 Evaluation against Research Questions

Fully adaptation of model-driven development method for the development service-based application is possible by using the abstraction concept. To review our results, we recapitulate the research questions posed in Section 1.3 that was broken down into seven sub research questions posed in Section 2.3.

1. *Q1: What models should we use to present run-time services for different service users?*

A model is used to represent something real that exists in run-time environment. In this case, a model is only a representation of a service. It can use different notations or symbols. For example, in PMG-pro, services can be presented using UML classes, SoaML Participants, SCA components and ARCTIS building blocks. The important thing is that any symbols are connected to source code that implements the concept of the used symbols or notations. The source code is used for the purpose of code generation. More explanation about this has been presented in Section 4.2.3.

2. *Q2: How can the presentation of run-time services into service models can be done?*

The presentation of run-time services can be done by employing their service descriptions. Current available service descriptions use XML to describe services, for example XML UPnP service description. We consider that XML is a language, therefore an XML UPnP service description is a model. The service models that are mentioned above (i.e. UML classes, SoaML Participants, SCA components, ARCTIS building blocks) are obviously a model.

The presentation of run-time services is done by applying a Model-to-Model transformation. The actual processing is done by a Text-to-Text transformation. In the case of transforming a UPnP service description into an ARCTIS building block, the transformation of XML (text) into XMI (text) is done. More detail about the transformation process of service description into service models has been presented in Section 4.2.

3. *Q3: Which service description technology should we use to describe services?*

Although this is a very important question in the field of software service engineering, we did not find any solution to answer the question. However, from the design-time perspective we consider that a service description must include information about service categorization. Within the PMG-pro, this

information will help the construction of hierarchical service models in the service library.

A service description should also describe states. With this information, any system can be modeled more accurately either as an event-based or request-reply system.

A proposal of a service description is presented in Section 4.2.2. The conceptual service description itself is presented in Figure 4.3. We propose to extend to UPnP service description for describing embedded services in the Internet of Services. A case study of using UPnP service description for building an event-based system is presented in Section 5.5.2.

4. *Q4: Which modeling languages should be used to model service-based applications?*

Any modeling language can be used to model service-based applications. The requirement is that each notation or symbol (instance of the language) is connected to source code that implements the concept of the notation or symbol. This can be done by developing different service presenters and abstractors that transform run-time services into service models in a hierarchical manner. With these service models, service integrators can model new service-based application at different abstraction levels, while the code generator uses the connected code to generate tailored source code.

PMG-pro focuses on a presentation of existing services using notations or symbols that conform to existing modeling languages. With this, software developers can use existing modeling languages to specify software system. Therefore, the semantics issues are depending on the chosen modeling languages. A relevant solution to this has been presented in Section 4.3.

5. *Q5: To what extent should device capability and configuration be included in the design-time (modeling step)?*

Using ontology, we propose to construct a hierarchical service model. The highest level of the hierarchy is the most abstract model, while the lowest level is the less abstract. All information about device capabilities and configurations are given at different levels in the hierarchy service models. The lowest level includes the complete information about the device capability and information. It is up to the service developers to choose in which level of the service models that will be used to model a service-based application. A relevant solution of this has been presented in Section 4.3.

6. *Q6: How can code generation from a high-level model of service-based application be automated?*

In PMG-pro, basically, the code generation will be handled by the built-in code generator of the chosen modeling languages. However, for the purpose of proof-of-concept of the providing step, PMG-pro has also its own code generator.

To generate code from the structure part PMG-pro uses the instantiation concept. For example, from a building block, object instantiations are used. To generate code from the behavior part, PMG-pro adapts the generation method presented in [10]. We have presented this in case study 3.

7. *Q7: Should we add a platform model or at least some platform patterns?*

The most important key of achieving an automated code generation is that every service model relates to code that implements service invocations. This code can be seen as a proxy to the real service resides in service providers. PMG-pro constructs these pairs (model-code) in a hierarchical manner.

With the constructed platform models, code can be generated from service models at any level of the service hierarchy. In other words, the platform information is already generated during the abstraction process and therefore it can be used for code generation of all possible platforms during the code generation step. However, templates are still required. The templates are constructed with a pattern following the target platforms.

6.3 Evaluation against Related Work

Service composition is gaining importance, as it can produce composite services with features not present in the individual basic services. Several research projects have been conducted with regards to the service creations by composition in the context of Future Internet. Those projects proposed different solutions (i.e. methods, languages, etc.) for the service creation from requirement capturing for service specification, service modeling, service design, service implementation, service deployment, and service maintenance. In this evaluation, firstly, we will present four related research projects and secondly, we will evaluate the PMG-pro method regarding their proposed methodologies.

6.3.1 Related Work

During 2005 - 2010, several research projects have been done in the area of service-based applications (within the European Commission ICT Research in FP7). Examples of such research project are the SeCSE project [103], the SODIUM project [104], the ASG project [31] and the ISIS project. Different modeling tools for service creation have also been developed. We will first present a short overview of the project mentioned above, including their proposed methods. Then we will evaluate them against PMG-pro.

6.3.1.1 ISIS

As it has been presented in earlier, the thesis work was done in the context of the ISIS project [110]. One of the ISIS results is ARCTIS. In ARCTIS, modeling is based on building blocks on the library. The service library contains all information about the services, such as its description, properties, capabilities, and run-time behaviors. The library is used by the service developer as the main component to build a new service.

An ARCTIS building block is constructed from three layers:

- Java code describes detailed behavior of operations on data and APIs.
- UML activities describe in which order operations are executed and how events arriving from different sources are synchronized.
- UML state machines (so-called ESMs) describe contracts that define in which context a building block may be used.

A service-based application can be specified as a building block composition. For the composition, UML activity diagram is used. With this, service engineers can think and understand the complete behavior of a specification, not only several scenarios. This has the benefit that complete specifications can be analyzed by model checking, and that an implementation can be generated automatically

6.3.1.2 SeCSE

Service Centric Systems Engineering (SeCSE) [103] is one of the research projects working on the topic of service development. A project result of interest to the service engineering community at large is the SeCSE conceptual model which explains concepts that are important within the SeCSE project. The model clarifies

- The meaning of a service

- The difference between a service and its public description
- The distinction between simple vs composite and stateless vs stateful services
- The various actors that exploit, offer, and manage services
- The relevant aspects concerning service discovery, composition, publication, execution and monitoring.

SeCSE is creating methods, tools and techniques to enable service integrators and service providers to develop and use dependable services and service-centric applications more easily. Contributions are being made in the areas of service and systems specification, service discovery, service-centric systems architectures and service monitoring and management.

SeCSE has also developed a methodology for service engineering, tackling the need for significant service discovery activities both during the requirements engineering phase of a project and also at run-time, when service discovery is needed in order to decide whether to start re-negotiation and re-planning in order to optimize the overall quality of a service composition.

6.3.1.3 SODIUM

Service- Oriented Development in a Unified fraMework (SODIUM) is another research project that mainly addresses the need for standards-based integration of heterogeneous services. SODIUM aims to define and implement an open source middleware platform, tools and methodology as well as models and languages for composing heterogeneous services offered by diverse service providers in an open, unified and standards-based way. To support the development of service-based applications, SODIUM focuses on three main topics of research as follows:

- Set of Languages
This includes three languages; Visual Service Composition Language (VSCL) for designing service composition graphs at multiple levels of details, Unified Service Composition Language (USCL) to facilitate the invocation and composition of multiple types of services, and Unified Service Query Language (USQL)
- Visual Service Composition Suite comprising all tools necessary for constructing and analyzing a Visual service Composition Graph (VCG).
This includes the Visual Editor that enables the construction of a Visual Composition Graph (VCG) using the VSCL, a VSCL to USCL Translator which translates the VCG, expressed in VSCL, into USCL descriptions.

- Run Time Environment

This part comprises the components necessary for the execution of a VCG.

6.3.1.4 ASG

The Adaptive Service Grid (ASG) [31] project aims to build a platform which will perform and support a service delivery lifecycle, enabling provision and consumption of complex services. A key concept is that ASG will use semantic information about services to fulfill user requests.

ASG is developing an open platform to automate the interchange and composition of software and services via the internet. Bridging the worlds of Grid computing and service-oriented architectures, it is developing concepts, methods and tools for an open platform for adaptive services discovery, creation, composition, and enactment.

Service in ASG are thus described not only syntactically (the technical interface to invoke a service) but also semantically. These semantics are represented by semantic service specifications, which include functional and non-functional properties and rely on ontology from various industrial domains. ASG components can make use of this information about what a service does in addition to how it does it.

6.3.2 Discussion

Before we evaluate the PMG-pro method against the research projects mentioned above, firstly, we will evaluate the thesis contribution to the ISIS project. With respect to the ISIS method, the PMG-pro method supports for a fully round-trip service engineering cycle. This means that we can use the created services to build a bigger service-based application in an incremental manner. Figure 6.2 shows an illustration of how the PMG-pro method contributes to the whole ISIS method.

From the figure, it can be seen that two main contributions of the PMG-pro method to the ISIS project are a presentation mechanism of existing services described using XML into service models and an additional code generator. The service presentation mechanism is done in the presentation step of the PMG-pro method. With the presentation mechanism, it will be possible to have a fully-cycle of software development. With the additional code generation, it enriches the ISIS method.

One of the results of the ISIS project is ARCTIS. To evaluate how PMG-pro and ARCTIS are different, we start with a statement of that PMG-pro is a method

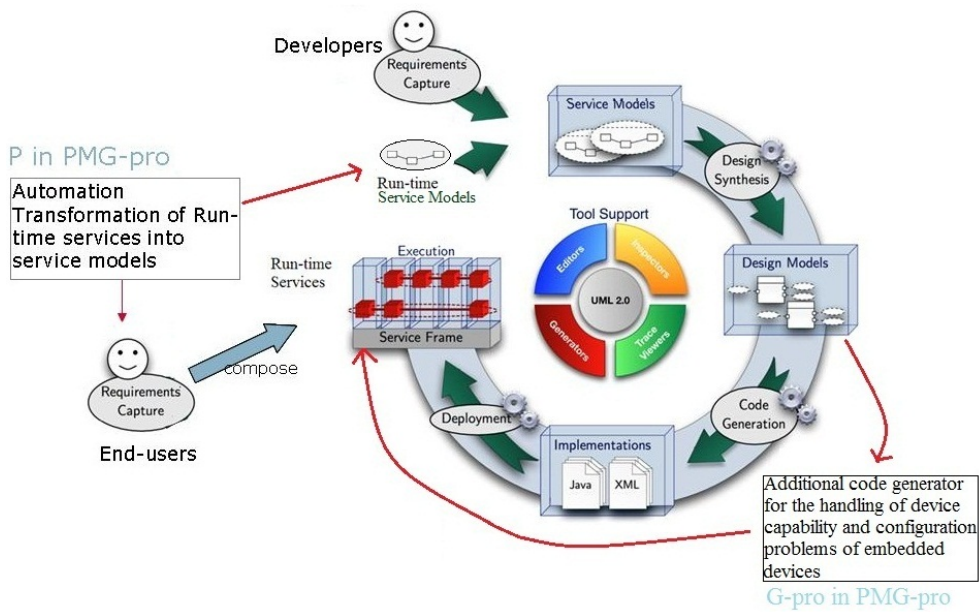


Figure 6.2: PMG-pro contribution to the ISIS method

while ARCTIS is a tool. The strong focus of PMG-pro is a method for the development of service-based applications using existing services by abstracting those services. From the perspectives of ARCTIS, the service presenter in PMG-pro can be seen as an automated tool for abstracting existing services into building blocks in the ARCTIS's library, since the ARCTIS is based on building blocks on the library [65, 53, 57]. As far as the author understood, a building block in ARCTIS is defined/developed/ modeled manually, and in an incremental manner. This includes the creation of code for implementing the building block functionality. Thus, PMG-pro enriches ARCTIS.

As mentioned earlier, PMG-pro consist of three parts; presentation, modeling and code generation. The modeling and code generation parts are already part of ARCTIS such that PMG-pro can use ARCTIS to model a service-based application and ARCTIS's code generator to generate code. ARCTIS does not manage the presentation part that automatically presents visually existing services. With PMG-pro, it will be possible to develop/specify new service by composing existing services in ARCTIS, without any effort to manually create building blocks representing the existing services. In this case, PMG-pro extends ARCTIS. That is the presentation part of the PMG-pro.

With regard to modeling part, it is necessary to note that PMG-pro is not only support ARCTIS, but also other modeling tools by considering that we can create different service presenters to the target tools. We have mentioned that in addition to

ARCTIS, the proof-of-concept tool of PMG-pro supports also UML Rational Rose. Other modeling tools can also be supported for example Objectteering for SoAML. In this case, PMG-pro can be seen as an ARCTIS user.

Both ARCTIS and PMG-pro can be used for high-level modeling. With the concept of blocks inside a block, ARCTIS supports for high-level modeling. In PMG-pro, a hierarchical service model is used. Service models are classified into different abstraction levels. The less the level service models, the platform dependent a service model would be. With the hierarchical concept, PMG-pro users can model a service-based application at different abstraction levels. It is up to the developers to specify in which level a model will be specified.

With regard to code generation PMG-pro and ARCTIS are also different. Each building block comes with its code. There are three cases of building blocks:

- a. building blocks for existing services - they get code from the abstractor,
- b. composition building blocks - they get their code from the underlying code in composed blocks.
- c. building blocks of aggregated services in the ontology. They get their code according to the description in 4.5.2.3.

Therefore the only place where the code is directly connected to the entity is case a, the rest is generated. In this case, ARCTIS handles only case a and b, and PMG-pro adds case c. In addition to an automated transformation of run-time services into service models and code, PMG-pro handles also high-level models concept by using the concept of service aggregation, case 3. PMG-pro uses the concept of ontology to manage the levels of the service models. The code generator in PMG-pro will generate required code for model execution.

There are two possible ways of code generation in PMG-pro. The first is to generate code for possible aggregated services in the ontology. All possible device configurations would also support run-time adaptation in case of services are removed or new services appear. The second possible solution is to generate code only for the present (specific) services. This solution requires information about what devices are available at runtime. This means code cannot be generated before the platform is used, which essentially means code generation on-demand. This is how the separation between the functionality of a problem and the access functions to particular run-time environments is realized in practice. That is, by generating code using the two possible ways mentioned above.

However, it is also necessary to note that PMG-pro can use built-in code generators of the used modeling tools. For example, if we have chosen ARCTIS as a target

tool at the presentation step, we can use the ARCTIS code generator to generate the constructed model. In this case, PMG-pro is only used to present existing services into ARCTIS building blocks. PMG-pro facilitates the ARCTIS users to be able to use existing services in an automatic manner.

As a summary, three important points of how PMG-pro differs with ARCTIS are; the service presentation, the use of ontology to construct the hierarchical service models, and the provisioning of code generation.

For the evaluation of the PMG-pro against the research project mentioned earlier, we will only focus on four issues; service presentation, modeling languages, service descriptions, and handling device capabilities and configuration. In particular, we will focus on how they solve these issues with regard to the sub questions presented in Section 2.3.

One thing that makes PMG-pro different from the projects mentioned above (SeCSE, ASG and SODIUM) is that, while they proposed new frameworks or/and new modeling languages for the development of service-based applications, PMG-pro uses the existing ones (e.g., modeling languages). PMG-pro focuses on an automated abstraction/presentation of existing services. Depending on the abstractor/presenter, any graphical presentation conforming to a modeling language can be used to present a service. We construct platform information by maintaining the relation between service models and their code (i.e., their real implementations). This is not the case of SeCSE, ASG and SODIUM. They have also developed their own framework for executing the composed services. Furthermore, SODIUM developed its own domain-specific languages.

The fact that different perspectives may have different definitions of a service means that the definition of service composition may also be different. PMG-pro considers that a service is just a kind model of a software unit. With this definition, a services composition can be done in a similar way as a composition of software units that normally is done at design-time using bottom-up approaches. In contrast, the three research projects mentioned above do not use this perspective.

For the composition of service component models (i.e., software units), composition techniques, and composition languages are required[3]. For the composition languages, there is no particular language that is supposed to be a language for the service compositions. In Web service context, the Web Service Business Process Execution Language (WS-BPEL) [82] and the Web Services Choreography Description Language (WS-CDL) [37] can be considered as a composition language. Within the OMG context, the Service-oriented architecture Modeling Language (SoaML) [79] is another example of composition languages. Also in the Web

services context, services orchestration and choreography are well-known service composition techniques. In the context of Service Component Architecture (SCA) [40], wiring can also be considered as a type of composition techniques. While PMG-pro is language-independent, the three research projects mentioned above seem to promote their own languages.

Presenting software functionality into abstract graphical representation has also been studied by other researchers. For example, in [71] UML is used to model Web services. The main contributions of their method are conversion rules between UML and web services described by WSDL. However, their work focused only on Web services and did not think about how automated code generation can be achieved. In [113], software components are visualized using graphical notations that developers can easily understand. They use a picture of a real device to present a software component. The integration is done by simply connecting components graphically. Obviously, the approach is only applicable for a specific domain. While PMG-pro is considering of presenting all types of run-time services using graphical representation, the three research projects mentioned above do not consider an automated presentation of run-time services.

An important feature of PMG-pro is that different graphical representations can be used to represent services. This leads to different ways of specifying new service-based applications. For example, if run-time services are visually presented as UML classes, then we can use sequence diagram to specify a new service-based application.

PMG-pro focuses on design-time compositions. However, the PMG-pro method can be extended to support run-time compositions. Using the PMG-pro method it would be possible to generate service graphical representation that can be used by end-users (i.e., run-time composition). In the ISIS project for example [110], ICE, an end-user composition, has been developed. A service in ICE is presented as a puzzle with either one input or one output. A composition is done by connecting puzzles. By developing a service abstractor/presenter, an ICE puzzle for end-users and source code for implementing service invocations can be generated. The ICE puzzle then can be used by the end-users to model the composition (at run-time) while the source code is used by ICE to execute the composition.

MDD is considered effective if the transformation of abstract models to more detailed models is an automatic process. In [38], a method for an automatic transformation of flow-global choreography models (UML activity diagram) into localized choreography models (ARCTIS model) is proposed. The author proposes to use Attributed Graph Grammar System (AGG) as the graph transformation engine. Us-

ing this method, ARCTIS code generator is used to generate an executable code. In contrast, despite of built-on code generator, PMG-pro implements only a simple interpreter that reads and convert any connection of activity nodes (in the activity diagrams) into code, directly. This is of course a weak approach. However, it works for a simple activity diagram, where the automation of code generation can be achieved. For more complex activity diagram, PMG-pro can use built-in code generator.

From the other view, PMG-pro can be seen as a bottom-up service development as it starts from abstracting run-time services into different service models. Using these abstract service models, new services are specified. It does not start from high level models, goes to more detail models, and finally code (e.g. PIM \Rightarrow PSM \Rightarrow code, in terms of MDA), a top down approach. In term of model-driven development, PMG-pro includes a reverse engineering process, enabling a fully cycle of model-driven development method. This feature can not be found in SeCSE, SODIUM and ASG. They apply the top down approach.

SeCSE, SODIUM, ASG and ISIS provide a facility and method for building service-based application by composing existing pre-made services. However, providing the new functionality of a service-based application as a new service is not explicitly considered. Moreover, the three research projects mentioned above did not implicitly mention the used deployment method of service-based applications on small devices that have a big variability with regard to the device capability and configuration. Accordingly, they did not propose any solution to this issue.

As summary of the evaluation, we present Table 6.1 that shows a comparison of the PMG-pro method with the related work mentioned above.

Table 6.1: Evaluation - A comparison

	SecSE	SODIUM	ASG	PMG-pro
Modeling Languages	UML	VSCL and USCL ^a	UML	Developer's preference
Service Presentation	UML classes	VCG ^b	UML classes	Depending the chosen languages
Service Description	Web services	Web services	Web services	XML-based description
Handling device capability and configuration	-	-	-	Using the hierarchical service models in the library

^aSee Section 6.3.1.3

^bVisual Composition Graph, see Section 6.3.1.3

Chapter 7

Summary and Future Work

This chapter gives the summary of the thesis and future work.

7.1 Summary

This thesis addresses service engineering methods and tools for the creation of new services by composing pre-made services. We envision that all devices in the Internet of Things provide their functionalities as services. They can appear in any kind of software-based Internet services. At run-time, they may be hosted on small devices to powerful computing devices. Accordingly, referring to the software-oriented architectures, service-based applications can be developed by composing existing heterogeneous services that are hosted by autonomous and networked devices. Hence, a new service is created by providing the new composite functionality of the service-based application as a service.

The thesis proposes PMG-pro, a language-independent, bottom-up and model-driven method for the development of service-based applications. In PMG-pro, a service is defined as a model of a software unit. Thus, composing services can be done in the similar way as with software composition system. For this, a service is presented using a graphical service model and is connected to code that relates to the implementation of the service models. Thus, a connected code can be seen as a proxy to the real service that resides in a service provider.

In PMG-pro, a model of service-based applications is specified in collaboration of service models. The model specifies which services are included (used) and how they are interacting with each other. The used service models define the structure of the composite services while the interactions between the service models define the behavior of the service-based application. Here, PMG-pro facilitates the presentation of run-time service models at design-time, so that they can be used to specify

models of service-based application. This is done by an automatic transformation of a service description into a service model and code for service invocations, by employing existing service frameworks and APIs.

Different modeling languages can be used to present existing (pre-made) service at design-time. Accordingly, different programming languages also can be used to implement the code for service invocations. The important thing to remember is that we need to maintain the connection between the service models and the code for service invocations. Here, the code is a real implementation of the service model (a concept).

To demonstrate the PMG-pro method, parts of the method (the presentation and code generation) has been prototyped and three case studies have been performed. It was demonstrated how different modeling tools can be used for the development of service-based applications, where two modeling tools (ARCTIS and Rational Rose) were used.

We conclude that the use of model-driven development approach for the service creation using existing heterogeneous services, can only be succeed if run-time services can be presented as service models using notations/symbols that conforms to a modeling language. At the same time, the service models must be connected to the code that implements the service invocations. In this way, service developer can uses the graphical service models to specify a new service (a type of service-based application), while the code generator will select the appropriate code from the service library. To be able to do this, service frameworks and APIs for implementing the services invocations on device levels are required.

The PMG-prop method supports the three criteria of software development that have been presented in the introduction chapter, in the following ways:

1. **Exploring the reusability of existing services.**

Component reuse is a key point of a rapid development method of software systems. By developing automated service presenter/abstractors it would be possible to use existing services at design-time. This can be achieved transforming run-time service descriptions into service models at design-time.

In PMG-pro, the relation between service models and code is handled in the service library. Ontology is used to construct more abstract service models. With this, service composers can browse and compose services at different abstraction levels. If all existing services that might have been implemented using different technologies, are represented as graphical services models they can be used for developing composite services. Hence, the automation of service presentation enables the exploring the reusability of existing services.

2. **Handling device capability and configuration.**

In order to achieve the automation of code generation, information about the actual targeted platform (with a specific capabilities and configuration) is required. We have shown two ways of using the constructed platform models. The first is to generate code for possible aggregated services in the ontology. All possible device configurations would also support run-time adaptation in case of services are removed or new services appear. The second possible solution is to generate code only for the present (specific) services.

3. **Managing the complexity so that service designers are shielded from complexity.**

Abstraction, both by the use of component-oriented development and model-driven development approaches, has been used for many years as a mean of managing the complexity of software system and their development. By combining service-oriented system and model-driven development approaches, PMG-pro provides a method for software modeling at high abstraction levels, far away from the implementation details. Hence, service integrators are shielded from the complexity.

Moreover, PMG-pro is language-independent. Different notations conforming to selected modeling languages can be used to present services. Using the selected modeling languages and editors, software developers can specify new service-based applications using collaboration activities or sequence diagrams.

7.2 Future Work

PMG-pro supports for a rapid and automatic development composite services in the Internet of services. With its flexibility, PMG-pro can use different modeling techniques by considering that we can develop service presenters that present services to the tools that implement different modeling techniques. Two prototypes of service presenters for two different tools with two different modeling techniques have been prototyped. The first is ARCTIS using collaboration activities modeling techniques and the second one is Rational Rose using sequence diagrams modeling techniques. However, the prototypes that have been developed only support UML tools with XMI version 1.1 (Rational Rose) and XMI 2.1 (ARCTIS), and without compatibility issues checking. To be more widely used, extending the PMG-pro capability could be done.

The most important thing is developing an automated service categorization. Categorization enables the manageability of services, which can help with the service discoverability for the code generation. With this, a fully-automated and a technology-independent method for service creation in the Internet of Services will be in place. Moreover, the extension should be done for the following purposes:

1. Covering Different Existing Service Technologies

Service developers should be able to include different service description technologies as a consequence of different ways of implementing service-oriented architectures. Unfortunately, only a few service composition techniques and language support for non-Web Services technology, while there are available other service technologies that are potential to be included in the development of heterogenous service-based applications. For this, service presenters and abstractors for different services technologies should be developed.

In the prototype of PMG-pro, new functionalities of service-based application can be provided only as UPnP services. In the future we will extend to other service technologies such as Web services, DPWS services, OSGi services and any other service technologies.

2. Implementing more Service Presenters and Abstractors

A Text-to-Model (e.g., XML service description to UML) transformation is important key for the success of the use of model-driven approach for the development of composite services. On the other hand, different service presenters and abstractors are needed for different modeling language. Currently, the service presenter and abstractor supports only for two modeling languages (i.e., UML and ARCTIS). To be able to use the method for many modeling and programming languages, more service presenter and abstractor could be developed.

3. Providing an Automated Service Presentation for End-users

From the end-user's perspective, a service can be seen as provided functionality (i.e., at run-time) that can be used for their daily life. Since services are embedded on the devices, with this perspective, a services composition can be seen as a device composition. Here the service presentation in the PMG-pro can contribute to the service presentation of run-time service to end-users. An initial case study of presenting embedded services for end-users has been done and published in the *Proceeding of Conference on Information Technology and Electrical Engineering 2011 Yogyakarta, Indonesia*. The case study

is presented in Appendix E. However, since the case study was only for ICE, we plan to extend to other end-users service compositions environment.

REFERENCES

- [1] Ali Rezafard, Andras Vilmos, et.al. Internet of things : Strategic research roadmap, 2009. Available at <http://www.europa.eu/information>. Last accessed at 20/09/2011.
- [2] Rajeev Alur, Gerard J. Holzmann, and Doron Peled. An analyzer for message sequence charts. *Software - Concepts and Tools*, 17(2):70–77, 1996.
- [3] Uwe Assmann. *Invasive software composition*. Springer, April 2003.
- [4] Luciano Baresi, Elisabetta Di Nitto, and Carlo Ghezzi. Toward open-world software: Issue and challenges. *Computer*, 39:36–43, 2006.
- [5] Mariano Belaunde and Paolo Falcarin. Realizing an mda and soa marriage for the development of mobile services. In Ina Schieferdecker and Alan Hartman, editors, *Model Driven Architecture Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 393–405. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-69100-6_28.
- [6] H. Ben-Abdallah, S. Leue, University of Waterloo. Dept. of Electrical, and Computer Engineering. *Expressing and analyzing timing constraints in message sequence chart specifications*. Technical report (University of Waterloo. Dept. of Electrical and Computer Engineering). Dept. of Electrical and Computer Engineering, University of Waterloo, 1997.
- [7] Gérard Berry. *The foundations of Esterel*, pages 425–454. MIT Press, Cambridge, MA, USA, 2000.
- [8] Jean Bézivin, Grégoire Dupé, Frédéric Jouault, Gilles Pitette, and Jamal Ed-dine Rougui. First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In *Proceeding of 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.
- [9] Jean Bezivin, Slimane Hammoudi, Denivaldo Lopes, and Jouault Jouault. Applying mda approach for web service platform. In *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*, pages 58–70, Washington, DC, USA, 2004. IEEE Computer Society.
- [10] A.K. Bhattacharjee and R.K. Shyamasundar. Validated code generation for activity diagrams. In *Proceedings of Distributed Computing and Internet*

- Technology*, volume 3816/2005 of *LNCS*, pages 508–521. Springer Berlin / Heidelberg, 2005.
- [11] Rolv Braek. MDA in perspective. In *Proceedings of First European Workshop on Model Driven Architecture with Emphasis on Industrial Application*, Enschede, The Netherlands, March 2004. University of Twente.
- [12] Egon Brger, Alessandra Cavarra, and Elvinia Riccobene. An asm semantics for uml activity diagrams. In Teodor Rus, editor, *Algebraic Methodology and Software Technology*, volume 1816 of *Lecture Notes in Computer Science*, pages 293–308. Springer Berlin / Heidelberg, 2000. 10.1007/3-540-45499-3-22.
- [13] Manfred Broy, Christoph Hofmann, Ingolf Krüger, and Monika Schmidt. A graphical description technique for communication in software architectures. Technical Report TUM-I9705, Technische Univerität München, 1997.
- [14] Nicholas Carriero. An implementation of Linda for a NUMA Machine. *Parallel Computing*, 24(7):1005–1021, 1998.
- [15] Antonio Carzaniga, Elisabetta Di Nitto, David S. Rosenblum, and Alexander L. Wolf. Issues in supporting event-based architectural styles. In *Proceedings of the third international workshop on Software architecture*, page 1720, Orlando, Florida, United States, 1998. ACM.
- [16] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design of a scalable event notification service: Interface and architecture. Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado at Boulder, USA, 1998.
- [17] D. Chakraborty, Y. Yesha, and A. Joshi. A distributed service composition protocol for pervasive environments. In *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, volume 4, page 25752580 Vol.4, Atlanta, Georgia USA, 2004. IEEE.
- [18] Seung Mo Cho, Hyung Ho Kim, Sung Deok Cha, and Doo Hwan Bae. A semantics of sequence diagrams. *Information Processing Letters*, 84(3):125 – 130, 2002.
- [19] Franco Cicirelli, Angelo Furfaro, and Libero Nigro. Integration and interoperability between Jini services and Web services. In *IEEE International*

- Conference on Services Computing*, volume 0, pages 278–285, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
- [20] James Coplien, Daniel Hoffman, and David Weiss. Commonality and variability in software engineering. *IEEE Software*, 15:37–45, November 1998.
- [21] D. Cotroneo, C. Di Flora, and S. Russo. A Jini framework for distributed service flexibility. In *Proceedings of the 10th Euromicro conference on Parallel, distributed and network-based processing*, EUROMICRO-PDP’02, pages 109–116, Washington, DC, USA, 2002. IEEE Computer Society.
- [22] Lorcan Coyle, Steve Neely, Graeme Stevenson, Mark Sullivan, Simon Dobson, and Paddy Nixon. Sensor fusion-based middleware for smart homes. *International Journal of Assistive Robotics and Mechatronics*, 8(2):53–60, 2007.
- [23] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The JEDI Event-Based infrastructure and its application to the development of the OPSS WFMS. *IEEE Trans. Softw. Eng.*, 27(9):827850, 2001.
- [24] W. Damm and D. Harel. Lscs: Breathing life into message sequence charts. Technical report, Jerusalem, Israel, Israel, 1998.
- [25] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15:313–341, 2008. 10.1007/s10515-008-0032-x.
- [26] Christoph Eichner, Hans Fleischhack, Roland Meyer, Ulrik Schrimpf, and Christian Stehno. Compositional semantics for uml 2.0 sequence diagrams using petri nets. In Andreas Prinz, Rick Reed, and Jeanne Reed, editors, *SDL 2005: Model Driven*, volume 3530 of *Lecture Notes in Computer Science*, pages 1260–1263. Springer Berlin / Heidelberg, 2005. 10.1007/11506843-9.
- [27] Luc Engelen and Mark van den Brand. Integrating textual and graphical modelling languages. *Electron. Notes Theor. Comput. Sci.*, 253:105–120, September 2010.
- [28] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, August 2005.

- [29] Rik Eshuis and Roel Wieringa. A formal semantics for uml activity diagrams - formalising workflow models, 2001.
- [30] Rik Eshuis and Roel Wieringa. A formal semantics for UML Activity Diagrams - formalising workflow models. Technical Report TR-CTIT-01-04, Enschede, 2001.
- [31] Marcus Flehmig. Adaptive Service Grid (ASG) - toward an adaptive stateful service environment. In *Presented in Future Service Engineering meeting, Brussel*, 2005.
- [32] UPnP Forum. UPnP standard, 2003. Available at <http://www.upnp.org/>. Last accessed at 20/09/2011.
- [33] UPnP Forum. Mediarenderer : Device template version 1.01, 2010. Available at <http://upnp.org/specs/av/UPnP-av-MediaRenderer-v3-Device.pdf>. Last accessed at 20/09/2011.
- [34] UPnP Forum. Mediaserver: Device template version 1.01, 2010. Available at <http://upnp.org/specs/av/UPnP-av-MediaServer-v3-Device.pdf>. Last accessed at 20/09/2011.
- [35] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. Integrating formal methods with Model-Driven engineering. In *Proceedings of International Conference on Software Engineering Advances*, pages 86–92, Los Alamitos, CA, USA, 2009. IEEE Computer Society.
- [36] James Goodwill. *Apache Axis Live: A Web Services Tutorial*. Sourcebeat, December 2004.
- [37] Juan Jos Pardo et.al Gregorio Daz. Automatic translation of ws-cdl choreographies to timed automata. In Mario Bravetti, Lela Kloul, and Gianluigi Zavattaro, editors, *EPEW/WS-FM*, volume 3670 of *LNCS*, pages 230–242. Springer, 2005.
- [38] Fenglin Han, Surya B. Kathayat, Hien Nam Le, Rolv Brk, and Peter Herрман. Towards choreography model transformation via graph transformation. In *IEEE Conference on Software Engineering and Service Science, ICSESS 2011.*, Beijing, China, 07/2011 2011. IEEE, IEEE.
- [39] Zef Hemel, Lennart C. L. Kats, and Eelco Visser. Code Generation by Model Transformation. A Case Study in Transformation Modularity. In J. Gray,

- A. Pierantonio, and A. Vallecillo, editors, *Proceeding of International Conference on Model Transformation (ICMT 2008)*, volume 5063 of *LNCS*, page 183198. Springer, June 2008.
- [40] IBM. Service component architecture. <http://www.ibm.com/developerworks/library/specification/ws-sca/>, November 2006.
- [41] IBM. Business process execution language for web services version 1.1. <http://www.ibm.com/developerworks/library/specification/ws-bpel/>, February 2007.
- [42] ITU. Press release 15 february 2010 - International Telecommunication Union. Technical report, International Telecommunication Union, 2010.
- [43] ITU-TS. Recommendation z.120: Message sequence chart-annex b: Formal semantics of message sequence chart. Technical report, International Telecommunication Union, 1998.
- [44] ITU-TS. Recommendation z.120: Message sequence chart (msc). Technical report, International Telecommunication Union, 2004.
- [45] Sun Java. Getting started with SunSPOTs, 2008. Available at <http://www.sunspotworld.com/about.html>. Last accessed at 20/09/2011.
- [46] Michael Jeronimo and Jack Weast. *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, May 2003.
- [47] Shanshan Jiang. *Some Service Issues in Adaptable Service Systems*. PhD thesis, Norwegian University of Science and Technology, Department of Telematics, 2008.
- [48] Li JiZhe and Yuan YongJun. Research and Implementation of Lightweight ESB with Microsoft .NET. *Japan-China Joint Workshop on Frontier of Computer Science and Technology*, 0:455–459, 2009.
- [49] G. Kapitsaki, D.A. Kateros, I.E. Foukarakis, G.N. Prezerakos, D.I. Kaklamani, and I.S. Venieris. Service composition: State of the art and future challenges. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, page 15, Budapest, HUNgary, 2007. IEEE.
- [50] Jens Khner. *Expert .NET Micro Framework (Expert)*. APress, 2008.

- [51] Cédric Kiss. Composite capabilities/preference profiles: Structure and vocabularies 2.0. Technical report, W3C, 2007.
- [52] Satoshi Konno. Cyberlink for java programming guide v.1.3, 2005. Available at <http://www.cybergarage.org/twiki/bin/view/Main/CyberLinkForJava>. Last accessed at 20/09/2011.
- [53] Frank Alexander Kraemer, Rolv Bræk, and Peter Herrmann. Compositional Service Engineering with Arctis. *Teletronikk*, 105(2009.1), 2009.
- [54] Frank Alexander Kraemer and Peter Herrmann. Transforming Collaborative Service Specifications into Efficiently Executable State Machines. In Karsten Ehring and Holger Giese, editors, *Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*, volume 7 of *Electronic Communications of the EASST*. EASST, 2007.
- [55] Frank Alexander Kraemer, Peter Herrmann, and Rolv Bræk. Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In R. Meersmann and Z. Tari, editors, *Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA), 2006, Montpellier, France*, volume 4276 of *Lecture Notes in Computer Science*, pages 1613–1632. Springer-Verlag Heidelberg, 2006.
- [56] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Model-Driven Construction of Embedded Applications based on Reusable Building Blocks – An Example. In Attila Bilgic, Reinhard Gotzhein, and Rick Reed, editors, *SDL 2009*, volume 5719 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag Berlin Heidelberg, 2009.
- [57] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. *Journal of Systems and Software*, 82(12):2068–2080, December 2009.
- [58] Peter B. Ladkin and Stefan Leue. Interpreting message flow graphs. *Formal Aspects of Computing*, 7, 1995.
- [59] Xiaoshan Li, Zhiming Liu, and H. Jifeng. A formal semantics of uml sequence diagram. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 168 – 177, 2004.

- [60] J.M. Marquez, J. Alamo, and J.A. Ortega. Distributing OSGi services: The OSIRIS domain connector. In *Proceedings of the Fourth International Conference on Networked Computing and Advanced Information Management, 2008. NCM '08.*, volume 1, pages 341–346, 2008.
- [61] Mari Matinlassi. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. In *Proceedings of the 26th International Conference on Software Engineering (ICSE '04)*, pages 127–136, Washington DC, USA, 2004. IEEE Computer Society.
- [62] Doug McIlroy. Mass-Produced software components. In *Proceedings of the 1st International Conference on Software Engineering*, pages 98, 88, 1968.
- [63] Geir Melby and Rolv Braek. Delivery of convergent telecom services on J2EE platforms. In *Proceeding of International Conference on Intellegence in Service Delivery Networks*, 2004.
- [64] Stephen J. Mellor, Kendall Scott, and Dirk Weise. *MDA distilled*. Addison-Wesley, 2004.
- [65] Frank Alexander Kræmer. Arctis and Ramses: Tool suites for rapid service engineering. In *Proceedings of NIK 2007 (Norsk informatikkonferanse), Oslo, Norway*, Oslo, 2007. Tapir Akademisk Forlag.
- [66] Frank Alexander Kræmer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks*. PhD thesis, Norwegian University of Science and Technology, Trondheim, August 2008.
- [67] M. Merabti, P. Fergus, O. Abuelma'atti, H. Yu, and C. Judice. Managing distributed networked appliances in home networks. *Proceedings of the IEEE*, 96(1):185, 166, 2008.
- [68] Gail C. Murphy, David Notkin, and Kevin Sullivan. Software reflexion models: bridging the gap between source and high-level models. In *Proceedings of the 3rd ACM SIGSOFT symposium on Foundations of software engineering*, SIGSOFT '95, pages 18–28, New York, NY, USA, 1995. ACM.
- [69] Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley Professional, dimensions: 7-3/8x9-1/4 edition, May 2002.

- [70] Roy Grønmo and Birger Møller-Pedersen. From sequence diagrams to state machines by graph transformation. In Laurence Tratt and Martin Gogolla, editors, *Theory and Practice of Model Transformations*, volume 6142 of *LNCS*, pages 93–107. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-13688-7_7.
- [71] Roy Grønmo, David Skogan, Ida Solheim, and Jon Oldevik. Model-Driven web services development. In *Proceedings of International Conference on e-Technology, e-Commerce, and e-Services*, volume 0, pages 42–45, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
- [72] Josef Noll, Sarfraz Alam, and Mohammad M. R. Chowdhury. Integrating mobile devices into semantic services environments. In *Proceedings of the 2008 The Fourth International Conference on Wireless and Mobile Communications*, pages 137–143, Washington, DC, USA, 2008. IEEE Computer Society.
- [73] OASIS. OASIS reference model for service oriented architecture (SOA), 2006. Available at <http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf>. Last accessed at 20/09/2011.
- [74] OASIS. Devices profile for web services version 1.1. Technical report, OASIS, 2009.
- [75] Jon Oldevik, Tor Neple, Roy Grønmo, Jan Aagedal, and Arne-J. Berre. Toward standardised model to text transformations. In *Model Driven Architecture Foundations and Applications*, volume 3748/2005 of *LNCS*, pages 239–253. Springer Berlin / Heidelberg, 2005.
- [76] OMG. Model-Driven Architecture guide, version 1.0.1, June 2003. Available at <http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>. Last accessed at 20/09/2011.
- [77] OMG. CORBA Component Model specification, v4.0, 2004. Available at <http://www.omg.org/cgi-bin/doc?formal/06-04-01>. Last accessed at 20/09/2011.
- [78] OMG. Meta object facility (MOF) 2.0 Query/View/Transformation specification final adopted specification ptc/05-11-01, 2005. Available at <http://www.omg.org/docs/ptc/05-11-01.pdf>. Last accessed at 20/09/2011.

- [79] OMG. Service oriented architecture Modeling Language (SoaML) : Specification for the UML Profile and Metamodel for Services (UPMS), 2009. Available at <http://www.omg.org/cgi-bin/doc?ptc/09-04-01.pdf>. Last accessed at 20/09/2011.
- [80] OMG. Unified Modeling Language specification, version 2.2, 2009. Available at <http://www.omg.org/technology/documents/formal/uml.htm>. Last accessed at 20/09/2011.
- [81] OMG. Systems Modeling Language (SysML) 1.2, June 2010. Available at <http://www.omg.org/spec/SysML/1.2/PDF>. Last accessed at 20/09/2011.
- [82] Chun Ouyang, Eric Verbeek, Wil M. P van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Science of Computer Programming*, 67(2-3):162198, 2007.
- [83] Dimitri Papadimitriou. Future Internet: The Cross-ETP Vision Document. Technical Report Version 1.0, European Future Internet Assembly, FIA, 2009.
- [84] Andy D. Pimentel. The artemis workbench for system-level performance evaluation of embedded systems. *IJES*, 3(3):181–196, 2008.
- [85] Terry Quatrani. *Visual modeling with Rational Rose 2000 and UML (2nd ed.)*. Addison-Wesley Longman Ltd., Essex, UK, UK, 2000.
- [86] Daniel Retkowitz and Monika Pienkos. Ontology-based configuration of adaptive smart homes. In *Proceedings of the 7th workshop on Reflective and adaptive middleware*, ARM '08, pages 11–16, New York, NY, USA, 2008. ACM.
- [87] Ioanna Roussaki, Ioannis Papaioannou, Dimitrios Tsesmetzis, Julia Kantorovitch, Jarmo Kalaoja, and Remco Poortinga. Ontology based service modelling for composability in smart home environments. In Max Mhlhuser, Alois Ferscha, and Erwin Aitenbichler, editors, *Constructing Ambient Intelligence*, volume 11 of *Communications in Computer and Information Science*, pages 411–420. Springer Berlin Heidelberg, 2008. 10.1007/978-3-540-85379-4-47.
- [88] Johannes Sametinger. *Software engineering with reusable components*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

- [89] Sancho. Definition for the term (software) service, sector abbreviations and definitions for a telecommunications thesaurus oriented database, ITU-T, 2009. <http://www.itu.int/sancho>.
- [90] Pierre Audoin Consultants SAS. Economic and social impact of software and software based services. Technical report, Pierre Audoin Consultants SAS, 2010.
- [91] Selo Sulistyoy. An automated services presentation method for supporting end-user compositions. In *Proceeding of Proceeding of Conference on Information Technology and Electrical Engineering 2011*, volume 2. IEEE Press, 2010.
- [92] Selo Sulistyoy. Model-based approaches for the development of event-based systems using embedded services. In *Proceeding of 2nd International Conference on Computer Technology and Development (ICCTD)*, pages 591 – 596, 2010.
- [93] Selo Sulistyoy. Presenting reusable service models in model-driven service engineering. In *Proceeding of International Conference on Future Information Technology - ICFIT 2010*, volume 2. IEEE Press, 2010.
- [94] Selo Sulistyoy and Andreas Prinz. Model-Driven development approach for providing smart home services. In *Ambient Assistive Health and Wellness Management in the Heart of the City*, volume 5597/2009 of LNCS, pages 274–277. Springer Berlin / Heidelberg, Tours, France, 2009.
- [95] Selo Sulistyoy and Andreas Prinz. Recursive modeling for completed code generation. In *Proceedings of 1st Workshop on Behavior Modeling in Model-Driven Architecture*, volume 1, pages 1–7, Enschede, The Netherlands, 2009. ACM New York, NY, USA.
- [96] Selo Sulistyoy and Andreas Prinz. Model-driven approaches for service-based applications development. In *Proceeding of 5th International Conference on Software and Data Technologies (ICSOFT 2010)*, volume 1, Piraeus, Greece, 2010. SciTePress. Portugal.
- [97] Selo Sulistyoy and Andreas Prinz. PMG-pro: a model-driven method for the development of service-based applications in a heterogeneous services environment. In *Proceeding of IEEE International Conference on Software Engineering and Service Sciences (ICSESS), 2010*, volume 5, pages 111–114, Beijing, China, 2010. IEEE Press.

- [98] Selo Sulistyono and Andreas Prinz. PMG-pro: A model-driven method for the development of service based applications. In *Proceeding of 15th International Conference on System Design Languages: Integrating system and software modeling*, volume 7083/2011 of LNCS, pages 136–151. Springer Berlin / Heidelberg, Toulouse, France, 2011.
- [99] Yu Sun, Zekai Demirezen, Marjan Mernik, Jeff Gray, and Barrett Bryant. Is my DSL a modeling or programming language? In *Proceedings of 2nd International Workshop on Domain-Specific Program Development (DSPD)*, Nashville, Tennessee, 2008.
- [100] York Sure. The internet of services. In *International Conference on Ontologies, DataBases, and Applications of SEMantics (ODBASE2007)*, 2007.
- [101] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming (2nd Edition)*. Addison-Wesley Professional, 2 edition, November 2002.
- [102] The Open Mobile Alliance Team. User agent profile 2.0. Technical report, Open Mobile Alliance, 2007.
- [103] The SeCSE Team. Designing and deploying service-centric systems: the SeCSE way. In *Proceedings of the Service Oriented Computing: a look at the Inside (SOC @Inside'07)*, 2007.
- [104] Simela Topouzidou. Service oriented development in a unified framework (SODIUMCSE). Deliverable CD-JRA-1.1.2, SODIUM Consortium, May 2007.
- [105] International Telecommunication Union. The internet of things - executive summary, 2005. Available at <http://www.itu.int/osg/spu/publications/internetofthings/InternetofThingssummary.pdf>. Last accessed at 20/09/2011.
- [106] Willem-Jan van den Heuvel, Olaf Zimmermann, and et.al. Software service engineering: Tenets and challenges. In *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, PESOS '09*, pages 26–33, Washington, DC, USA, 2009. IEEE Computer Society.
- [107] V. Vitolins and A. Kalnins. Semantics of uml 2.0 activity diagram for business modeling by means of virtual machine. In *EDOC Enterprise Computing Conference, 2005 Ninth IEEE International*, pages 181 – 192, sept. 2005.

- [108] J. White, D.C. Schmidt, E. Wuchner, and A. Nechypurenko. Automating product-line variant selection for mobile devices. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 129–140, 2007.
- [109] Chao-Lin Wu, Chun-Feng Liao, and Li-Chen Fu. Service-oriented smart-home architecture based on OSGi and mobile-agent technology. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(2):193–205, 2007.
- [110] Xiaomeng Su, Reidar Martin Svendsen, et.al. *Description of the ISIS Ecosystem Towards an Integrated Solution to Internet of Things*. Telenor Group Corporate Development, 2010.
- [111] Jingjing Xu, Yann-Hang Lee, Wei-Tek Tsai, Wu Li, Young-Sung Son, Jun-Hee Park, and Kyung-Duk Moon. Ontology-based smart home solution and service composition. In *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, pages 297–304, may 2009.
- [112] William Yeager and Joseph Williams. Secure peer-to-peer networking: The jxta example. *IT Professional*, 4:53–57, 2002.
- [113] Kostyantyn Yermashov. *Software Composition with Templates*. PhD Thesis, De Montfort University, UK, 2008.
- [114] L. Youseff, M. Butrico, and D. Da Silva. Toward a Unified Ontology of Cloud Computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov 2008.
- [115] Elmar Zeeb, Andreas Bobek, Hendrik Bohn, and et.al. WS4D: SOA-Toolkits making embedded systems ready for web services. In *Proceedings of Second International Workshop on Open Source Software and Product Lines*, page 3342, Limerick, 2007. ITEA.
- [116] Simon Znaty and Jean-Pierre Hubaux. Telecommunications services engineering: Principles, architectures and tools. In *Object-Oriented Technologys*, volume 1357/1998 of *LNCS*, pages 3–10. Springer Berlin / Heidelberg, 1998.

Appendix A

List of Publications

During the period of thesis work, together with the supervisor, the author of this thesis has published several publications in conference proceedings.

1. Sulisty, S., **An Automated Services Presentation Method for Supporting End-user Compositions**. In Proceeding of International Conference on Information Technology and Electrical Engineering, July 28th, 2011, Yogyakarta, Indonesia. ISBN: 979-97956-0-7 (2011)
2. Sulisty, S., Prinz, A. **PMG-pro: A Model-driven Development Method of Service-based Applications**. In Proceeding of 15th International Conference on System Design Languages: *Integrating system and software modeling*, Toulouse, France. LNCS Volume 7083, pp. 136–151. Springer, Heidelberg (2011).
3. Sulisty, S., Prinz, A. **Model-driven Approaches for Service-based Applications Development**. In Proceeding of 5th International Conference on Software and Data Technologies (ICSOFT 2010), July 22-24,2010, Piraeus, Athen, Greece. SciTePress. Portugal. ISBN: 978-989-8425-23-2 (2010)
4. Sulisty, S., Prinz, A. **PMG-pro: A Model-driven Method for the Development of Service-based Applications in a Heterogeneous Services Environment**. In: Proceeding of IEEE International Conference on Software Engineering and Service Sciences (ICSESS), July 16-18, Beijing, China. IEEE Press. ISBN: 978-1-4244-6053-3 (2010)
5. Sulisty, S., Prinz, A. **Systems, Models and Languages**. In: Proceeding of IEEE 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE 2010), August 20-22, Chengdu, China. IEEE Press. ISBN: 978-1-4244-6540-8, ISSN: 2154-7491 (2010)

6. Sulistyoy, S. 2010. **Presenting Reusable Service Models in Model-driven Service Engineering**. In: Proceeding of 2010 International Conference on Future Information Technology (ICFIT), 14-15 December 2010. ISBN: 978-1-4244-8370-9. (2010)
7. Sulistyoy, S. **Model-based Approaches for the Development of Event-based Systems Using Embedded Services**. In: Proceeding of 2nd International Conference on Computer Technology and Development, Cairo, Egypt. November 2-4, ISBN: 978-1-4244-8843-8. (2010)
8. Sulistyoy, S. and Prinz, A. **Recursive modeling for completed code generation**, In: Proceedings of the 1st Workshop on Behaviour Modelling in Model-Driven Architecture, Enschede, The Netherlands, ed: ACM New York, USA ISBN: 978-1-60558-503-1 (2009)
9. Sulistyoy, S., and A. Prinz, **A Model-Driven Development Approach for Providing Smart Home Services**, In: Proceeding of Ambient Assistive Health and Wellness Management in the Heart of the City. LNCS Volume 5597/2009, ed: Springer Berlin / Heidelberg, 2009, pp. 274-277. ISBN: 978-3-642-02867-0 (2009)

Additionally, the author has been collaborated with another PhD student focusing on the performance of security algorithms on small devices. The author took part on the implementation of software embedded systems. The collaboration has produced two publications.

- Yi Ren, Vladimir Oleshchuk, Frank Y. Li, and Selo Sulistyoy, SCARKER: a Sensor Capture Resistance and Key Refreshing Scheme for Mobile WSNs. In Proceeding of the 36th IEEE Conference on Local Computer Networks (LCN), Bonn, Germany, 4-7 October 2011 (2011).
- Yi Ren, Vladimir Oleshchuk, Frank Y. Li and Selo Sulistyoy, "FoSBaS: A Bi-directional Secrecy and Collusion Resilience Key Management Scheme for BANs". Accepted for publication in IEEE Wireless Communications and Networking Conference (WCNC 2012), Paris, France, 1-4 April 2012 (2012).

Appendix B

UPnP Light Service Description

B.1 Device Description.xml

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <device>
    <deviceType>urn:schemas-upnp-org:device:BinaryLight:1</deviceType>
    <presentationURL></presentationURL>
    <friendlyName>Light (AUB30307-LAD830)</friendlyName>
    <manufacturer>Intel Corporation</manufacturer>
    <manufacturerURL>http://www.intel.com</manufacturerURL>
    <modelDescription>Software Emulated Light Bulb</modelDescription>
    <modelName>Intel CLR Emulated Light Bulb</modelName>
    <modelName>XPC-L1</modelName>
    <modelURL>http://www.intel.com/xpc</modelURL>
    <UDN>uuid:c67c29a2-671f-41e7-8152-eef45096bad4</UDN>
    <iconList>
      <icon>
        <mimetype>image/png</mimetype>
        <width>32</width>
        <height>32</height>
        <depth>32</depth>
        <url>/icon.png</url>
      </icon>
      <icon>
        <mimetype>image/jpg</mimetype>
        <width>32</width>
        <height>32</height>
        <depth>32</depth>
        <url>/icon.jpg</url>
      </icon>
    </iconList>
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:service:DimmingService:1</serviceType>
        <serviceId>urn:upnp-org:serviceId:DimmingService.$0001</serviceId>
        <SCPDURL>_urn:upnp-org:serviceId:DimmingService.0$001_scpd.xml</SCPDURL>
      </service>
    </serviceList>
  </device>
</root>
```

```

    <controlURL>_urn:upnp-org:serviceId:DimmingService.$0001_control</controlURL>
    <eventSubURL>_urn:upnp-org:serviceId:DimmingService.$0001_event</eventSubURL>
  </service>
  <service>
    <serviceType>urn:schemas-upnp-org:service:SwitchPower:1</serviceType>
    <serviceId>urn:upnp-org:serviceId:SwitchPower.$0001</serviceId>
    <SCPDURL>_urn:upnp-org:serviceId:SwitchPower.$0001_scpd.xml</SCPDURL>
    <controlURL>_urn:upnp-org:serviceId:SwitchPower.$0001_control</controlURL>
    <eventSubURL>_urn:upnp-org:serviceId:SwitchPower.$0001_event</eventSubURL>
  </service>
</serviceList>
</device>
</root>

```

B.2 Dimming Service Description.xml

```

<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>GetLoadLevelStatus</name>
      <argumentList>
        <argument>
          <name>RetLoadLevelStatus</name>
          <direction>out</direction>
          <relatedStateVariable>LoadLevelStatus</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetMinLevel</name>
      <argumentList>
        <argument>
          <name>MinLevel</name>
          <direction>out</direction>
          <relatedStateVariable>MinLevel</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SetLoadLevelTarget</name>
      <argumentList>
        <argument>
          <name>NewLoadLevelTarget</name>
          <direction>in</direction>
          <relatedStateVariable>LoadLevelTarget</relatedStateVariable>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="yes">
      <name>LoadLevelStatus</name>

```

```

        <dataType>ui1</dataType>
        <allowedValueRange>
            <minimum>0</minimum>
            <maximum>100</maximum>
        </allowedValueRange>
    </stateVariable>
<stateVariable sendEvents="no">
    <name>MinLevel</name>
    <dataType>ui1</dataType>
</stateVariable>
<stateVariable sendEvents="no">
    <name>LoadLevelTarget</name>
    <dataType>ui1</dataType>
    <allowedValueRange>
        <minimum>0</minimum>
        <maximum>100</maximum>
    </allowedValueRange>
</stateVariable>
</serviceStateTable>
</scpd>

```

B.3 Switch Power Service Description.xml

```

<?xml version="1.0" encoding="utf-8"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <actionList>
        <action>
            <name>GetStatus</name>
            <argumentList>
                <argument>
                    <name>ResultStatus</name>
                    <direction>out</direction>
                    <relatedStateVariable>Status</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>SetTarget</name>
            <argumentList>
                <argument>
                    <name>newTargetValue</name>
                    <direction>in</direction>
                    <relatedStateVariable>Target</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
    </actionList>
    <serviceStateTable>
        <stateVariable sendEvents="yes">
            <name>Status</name>
            <dataType>boolean</dataType>
        </stateVariable>
        <stateVariable sendEvents="no">

```

```
    <name>Target</name>
    <dataType>boolean</dataType>
  </stateVariable>
</serviceStateTable>
</scpd>
```

Appendix C

XMI Representation of ARCTIS Building Blok of UPnP Light

C.1 ARCTIS building block

The ARCTIS building block of UPnP Light service is shown in Figure C.1. All UPnP actions are transformed into parameter inputs while the State Variables, which has *invented is true*, are transformed into parameter outputs. In addition, all get actions are also transformed into parameter output event. The parameter is taken from the name of argument type of 'out'.



Figure C.1: An ARCTIS building block is used to represent run-time UPnP Light services, see also Figure 4.7

C.2 The XMI file

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:arctis="http://schemas/arctis/_WFTqUHjGEdyxtoUMycQn9Q/10"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:graphics="http://schemas/graphics/_gd9EwIuAEdytfOWtpACIqw/6"
```

```

xmlns:uml="http://www.eclipse.org/uml2/2.1.0/UML"
xsi:schemaLocation="http://schemas/arctis/_WFTqUHjGEdyxtoUMycQn9Q/10
pathmap://ARCTIS_PROFILES/arctis.profile.uml#_WFTqUXjGEdyxtoUMycQn9Q
http://schemas/graphics/_gd9EwIuAEdytfOWtpACIqw/6 pathmap://RAMSES_GRAPHIC_PROFILES/
no.ntnu.item.ramses.graphicprofile.uml#_geG1wIuAEdytfOWtpACIqw">
  <uml:Package xmi:id="_sPnt8LymEd6VCq2W8FCNPQ" name="no.uia.isis.runtimeupnp.light">
    <packagedElement xmi:type="uml:Activity" xmi:id="_sPgo4LymEd6VCq2W8FCNPQ"
name="UPnPLight" classifierBehavior="_sd0KoLymEd6VCq2W8FCNPQ"
partition="_sPgo4bymEd6VCq2W8FCNPQ">
      <ownedBehavior xmi:type="uml:StateMachine"
xmi:id="_sd0KoLymEd6VCq2W8FCNPQ" name="UPnPLight">
        <region xmi:id="_sd97oLymEd6VCq2W8FCNPQ" name="&lt;region>">
          <subvertex xmi:type="uml:Pseudostate"
xmi:id="_seHFkLymEd6VCq2W8FCNPQ" name="&lt;initial>" />
            <subvertex xmi:type="uml:State" xmi:id="_4M_HgLymEd6VCq2W8FCNPQ" name="on" />
              <subvertex xmi:type="uml:State" xmi:id="_cAMFILymEd6VCq2W8FCNPQ" name="off" />
                <transition xmi:id="_4MYqkLymEd6VCq2W8FCNPQ" name="setTarget/Status"
target="_cAMFILymEd6VCq2W8FCNPQ" source="_4M_HgLymEd6VCq2W8FCNPQ" />
                <transition xmi:id="_b_vZMLymEd6VCq2W8FCNPQ" name="setTarget/Status"
target="_4M_HgLymEd6VCq2W8FCNPQ" source="_cAMFILymEd6VCq2W8FCNPQ" />
                <transition xmi:id="_0haZkLysEd6VCq2W8FCNPQ" name="setTarget/Status"
target="_4M_HgLymEd6VCq2W8FCNPQ" source="_4M_HgLymEd6VCq2W8FCNPQ" />
                <transition xmi:id="_1HRzMLysEd6VCq2W8FCNPQ" name="setTarget/Status"
target="_cAMFILymEd6VCq2W8FCNPQ" source="_cAMFILymEd6VCq2W8FCNPQ" />
              </region>
            </ownedBehavior>
          <ownedOperation xmi:id="_58oDsbynEd6VCq2W8FCNPQ" name="getArgument" />
          <ownedOperation xmi:id="_dSFqgbyqEd6VCq2W8FCNPQ" name="getStatus" />
          <ownedOperation xmi:id="_gIxjwLyqEd6VCq2W8FCNPQ" name="setLoadLevelTarget" />
          <ownedOperation xmi:id="_hdQqcbyqEd6VCq2W8FCNPQ" name="setTarget" />
          <ownedOperation xmi:id="_ksY44LyqEd6VCq2W8FCNPQ" name="sendMinLevel" />
          <ownedOperation xmi:id="_u17UsbyqEd6VCq2W8FCNPQ" name="sendStatus" />
          <ownedOperation xmi:id="_xdRKYbyqEd6VCq2W8FCNPQ" name="sendLoadLevelStatus" />
            <ownedParameter xmi:id="_B301ILymEd6VCq2W8FCNPQ" name="setTarget"
type="_OKLj8LymEd6VCq2W8FCNPQ" isStream="true" />
            <ownedParameter xmi:id="_DGzJsLymEd6VCq2W8FCNPQ" name="setLoadLevelTarget"
type="_ytWZkLymEd6VCq2W8FCNPQ" isStream="true" />
            <ownedParameter xmi:id="_Eq718LymEd6VCq2W8FCNPQ" name="getMinLevel"
type="_ytWZkLymEd6VCq2W8FCNPQ" isStream="true" />
            <ownedParameter xmi:id="_GLpawLymEd6VCq2W8FCNPQ" name="getLoadLevelStatus"
type="_OKLj8LymEd6VCq2W8FCNPQ" isStream="true" />
            <ownedParameter xmi:id="_G-nIwLymEd6VCq2W8FCNPQ" name="getStatus"
type="_63LSIO_FEd6VUYZjXdLT1w" isStream="true" />
            <ownedParameter xmi:id="_WDb2ALyrEd6VCq2W8FCNPQ" name="Status"
direction="out" isStream="true" />
            <ownedParameter xmi:id="_XZ3dcLyrEd6VCq2W8FCNPQ" name="loadLevelStatus"
direction="out" isStream="true" />
            <ownedParameter xmi:id="_fbM1YEkuEd-L3Z940HHEpA" name="resultStatus"
direction="out" isStream="true" />
            <ownedParameter xmi:id="_kC1e4EkuEd-L3Z940HHEpA" name="minLevel"
direction="out" isStream="true" />
            <ownedParameter xmi:id="_mkqicEkuEd-L3Z940HHEpA" name="retLoadLevelStatus"
direction="out" isStream="true" />
          <variable xmi:id="_qVNC4LymEd6VCq2W8FCNPQ" name="target" type="_OKLj8LymEd6VCq2W8FCNPQ" />
          <variable xmi:id="_rjwbkLymEd6VCq2W8FCNPQ" name="status" type="_OKLj8LymEd6VCq2W8FCNPQ" />

          <variable xmi:id="_swxGYLymEd6VCq2W8FCNPQ"

```



```

name="loadLevelStatus" type="_ytWZk LynEd6VCq2W8FCNPQ"/>

<variable xmi:id="_swxGYLynEd6VCq2W8FCNPQ"
name="LoadLevelStatus" type="_ytWZk LynEd6VCq2W8FCNPQ"/>

<variable xmi:id="_uXJOULynEd6VCq2W8FCNPQ"
name="LoadLevelTarget" type="_ytWZkLynEd6VCq2W8FCNPQ"/>

<variable xmi:id="_v2zqQLynEd6VCq2W8FCNPQ"
name="MinLevel" type="_ytWZkLynEd6VCq2W8FCNPQ"/>

<node xmi:type="uml:ActivityParameterNode" xmi:id="_B2yTULynEd6VCq2W8FCNPQ"
name="setTarget" outgoing="_4paoALyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" type="_OKLj8LynEd6VCq2W8FCNPQ"
parameter="_B301ILynEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:ActivityParameterNode" xmi:id="_DFm24LynEd6VCq2W8FCNPQ"
name="setLoadLevelTarget" outgoing="_4LbB0LyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" type="_ytWZkLynEd6VCq2W8FCNPQ"
parameter="_DGzJsLynEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:ActivityParameterNode" xmi:id="_Ep5UIILynEd6VCq2W8FCNPQ"
name="getMinLevel" outgoing="_4aXfYLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" type="_ytWZkLynEd6VCq2W8FCNPQ"
parameter="_Eq718LynEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:ActivityParameterNode" xmi:id="_GKdH8LynEd6VCq2W8FCNPQ"
name="getLoadLevelStatus" outgoing="_38HdILyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" type="_OKLj8LynEd6VCq2W8FCNPQ"
parameter="_GLpawLynEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:ActivityParameterNode" xmi:id="_G-Kc0LynEd6VCq2W8FCNPQ"
name="getStatus" outgoing="_45g5gLyqEd6VCq2W8FCNPQ" inPartition="_sPgo4bymEd6VCq2W8FCNPQ"
type="_63LSIO_FEd6VUYZjXdLT1w" parameter="_G-nIwLynEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:CallOperationAction" xmi:id="_gInyLyqEd6VCq2W8FCNPQ"
name="o1" outgoing="_If4xcLyrEd6VCq2W8FCNPQ" incoming="_4LbB0LyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" operation="_gIxjwLyqEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:CallOperationAction" xmi:id="_hdQqcLyqEd6VCq2W8FCNPQ"
name="o2" outgoing="_I3buwLyrEd6VCq2W8FCNPQ" incoming="_4paoALyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" operation="_hdQqcbyqEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:CallOperationAction" xmi:id="_ksPH4LyqEd6VCq2W8FCNPQ"
name="o3" outgoing="_21HuUEkuEd-L3Z940HHEpA" incoming="_4aXfYLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" operation="_ksY44LyqEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:CallOperationAction" xmi:id="_u17UsLyqEd6VCq2W8FCNPQ"
name="o0" outgoing="_3TAvEEkuEd-L3Z940HHEpA" incoming="_45g5gLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" operation="_u17UsbyqEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:CallOperationAction" xmi:id="_xdRKYLyqEd6VCq2W8FCNPQ"
name="o4" outgoing="_2QjMgEkuEd-L3Z940HHEpA" incoming="_38HdILyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" operation="_xdRKYbyqEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:SendSignalAction" xmi:id="_7lBdgLyqEd6VCq2W8FCNPQ" name="s0"
outgoing="_ZLHYoLyrEd6VCq2W8FCNPQ" incoming="_I3buwLyrEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" signal="_CIfxkLyrEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:SendSignalAction" xmi:id="_FbWCYLyrEd6VCq2W8FCNPQ"
name="s1" outgoing="_vMo64CxQEd-TYfFYHHQxA" incoming="_If4xcLyrEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" signal="_EFXG4LyrEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:ActivityParameterNode" xmi:id="_WCZUMLyrEd6VCq2W8FCNPQ"
name="Status" incoming="_ZLHYoLyrEd6VCq2W8FCNPQ" inPartition="_sPgo4bymEd6VCq2W8FCNPQ"
parameter="_Wdb2ALyrEd6VCq2W8FCNPQ"/>
<node xmi:type="uml:ActivityParameterNode" xmi:id="_XYrKoLyrEd6VCq2W8FCNPQ"
name="LoadLevelStatus" incoming="_vMo64CxQEd-TYfFYHHQxA"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" parameter="_XZ3dcLyrEd6VCq2W8FCNPQ"/>

```

```

<node xmi:type="uml:ActivityParameterNode" xmi:id="_famvgEkuEd-L3Z940HHEpA"
name="ResultgetStatus" incoming="_3TAvEEkuEd-L3Z940HHEpA"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" parameter="_fbM1YEkuEd-L3Z940HHEpA"/>
<node xmi:type="uml:ActivityParameterNode" xmi:id="_kCPpAEkuEd-L3Z940HHEpA"
name="ResultgetMinLevel" incoming="_21HuUEkuEd-L3Z940HHEpA"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" parameter="_kC1e4EkuEd-L3Z940HHEpA"/>
<node xmi:type="uml:ActivityParameterNode" xmi:id="_mj67kEkuEd-L3Z940HHEpA"
name="ResultgetLoadLevelStatus" incoming="_2QjMgEkuEd-L3Z940HHEpA"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ" parameter="_mkqicEkuEd-L3Z940HHEpA"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_38HdILyqEd6VCq2W8FCNPQ" name="e0"
source="_GkdH8LynEd6VCq2W8FCNPQ" target="_xdRKYLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_4LbB0LyqEd6VCq2W8FCNPQ"
name="e1" source="_DFm24LynEd6VCq2W8FCNPQ" target="_gInywLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_4aXfYLyqEd6VCq2W8FCNPQ" name="e2"
source="_Ep5UIlYnEd6VCq2W8FCNPQ" target="_ksPH4LyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_4paoALyqEd6VCq2W8FCNPQ"
name="e3" source="_B2yTULynEd6VCq2W8FCNPQ" target="_hdQqcLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_45g5gLyqEd6VCq2W8FCNPQ"
name="e4" source="_G-Kc0LynEd6VCq2W8FCNPQ" target="_u17UsLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_If4xcLyrEd6VCq2W8FCNPQ"
name="e5" source="_gInywLyqEd6VCq2W8FCNPQ" target="_FbWCYLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_I3buwLyrEd6VCq2W8FCNPQ"
name="e6" source="_hdQqcLyqEd6VCq2W8FCNPQ" target="_71BdgLyqEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_ZLHYoLyrEd6VCq2W8FCNPQ"
name="e11" source="_71BdgLyqEd6VCq2W8FCNPQ" target="_WCZUMLyrEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ObjectFlow" xmi:id="_vMo64CxQEd-TYfFXyHHQxA"
name="e10" source="_FbWCYLyqEd6VCq2W8FCNPQ" target="_XYrKoLyrEd6VCq2W8FCNPQ"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_2QjMgEkuEd-L3Z940HHEpA"
name="e7" source="_xdRKYLyqEd6VCq2W8FCNPQ" target="_mj67kEkuEd-L3Z940HHEpA"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_21HuUEkuEd-L3Z940HHEpA"
name="e8" source="_ksPH4LyqEd6VCq2W8FCNPQ" target="_kCPpAEkuEd-L3Z940HHEpA"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<edge xmi:type="uml:ControlFlow" xmi:id="_3TAvEEkuEd-L3Z940HHEpA" name="e9"
source="_u17UsLyqEd6VCq2W8FCNPQ" target="_famvgEkuEd-L3Z940HHEpA"
inPartition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<group xmi:type="uml:ActivityPartition" xmi:id="_sPgo4bymEd6VCq2W8FCNPQ"
name="main" node="_B2yTULynEd6VCq2W8FCNPQ _DFm24LynEd6VCq2W8FCNPQ
_Ep5UIlYnEd6VCq2W8FCNPQ _GkdH8LynEd6VCq2W8FCNPQ _G-Kc0LynEd6VCq2W8FCNPQ
_gInywLyqEd6VCq2W8FCNPQ _hdQqcLyqEd6VCq2W8FCNPQ _ksPH4LyqEd6VCq2W8FCNPQ
_u17UsLyqEd6VCq2W8FCNPQ _xdRKYLyqEd6VCq2W8FCNPQ _71BdgLyqEd6VCq2W8FCNPQ
_FbWCYLyqEd6VCq2W8FCNPQ _WCZUMLyrEd6VCq2W8FCNPQ _XYrKoLyrEd6VCq2W8FCNPQ
_famvgEkuEd-L3Z940HHEpA _kCPpAEkuEd-L3Z940HHEpA _mj67kEkuEd-L3Z940HHEpA"
edge="_38HdILyqEd6VCq2W8FCNPQ _4LbB0LyqEd6VCq2W8FCNPQ _4aXfYLyqEd6VCq2W8FCNPQ
_4paoALyqEd6VCq2W8FCNPQ _45g5gLyqEd6VCq2W8FCNPQ _If4xcLyrEd6VCq2W8FCNPQ
_I3buwLyrEd6VCq2W8FCNPQ _ZLHYoLyrEd6VCq2W8FCNPQ _vMo64CxQEd-TYfFXyHHQxA
_2QjMgEkuEd-L3Z940HHEpA _21HuUEkuEd-L3Z940HHEpA _3TAvEEkuEd-L3Z940HHEpA"/>
</packagedElement>

```

```

<packagedElement xmi:type="uml:PrimitiveType" xmi:id="_ytWzkLynEd6VCq2W8FCNPQ" name="int"/>
<packagedElement xmi:type="uml:PrimitiveType" xmi:id="_OKLj8LynEd6VCq2W8FCNPQ" name="boolean"/>
<packagedElement xmi:type="uml:Signal" xmi:id="_CIfxkLyrEd6VCq2W8FCNPQ" name="Status"/>
<packagedElement xmi:type="uml:Signal" xmi:id="_EFXG4LyrEd6VCq2W8FCNPQ" name="LoadLevelStatus"/>
<packagedElement xmi:type="uml:PrimitiveType"
  xmi:id="_63LSIO_FEd6VUYZjXdLT1w" name="java.lang.String"/>
<packagedElement xmi:type="uml:SignalEvent" xmi:id="_1eX-sCxQEd-TYfFXyHHQxA" name="r0"/>
<profileApplication xmi:id="_sPW34LymEd6VCq2W8FCNPQ">
  <eAnnotations xmi:id="_sPW34bymEd6VCq2W8FCNPQ" source="http://www.eclipse.org/uml2/2.0.0/UML">
    <references xmi:type="ecore:EPackage" href="pathmap://ARCTIS_PROFILES
      /arctis.profile.uml#_WFTqUXjGEdyxtoUMycQn9Q"/>
  </eAnnotations>
  <appliedProfile href="pathmap://ARCTIS_PROFILES/arctis.profile.uml#_rE77YK4iEduZSuTiwKs5Lw"/>
</profileApplication>
<profileApplication xmi:id="_sPqZ4LymEd6VCq2W8FCNPQ">
  <eAnnotations xmi:id="_sPqZ4bymEd6VCq2W8FCNPQ" source="http://www.eclipse.org/uml2/2.0.0/UML">
    <references xmi:type="ecore:EPackage"
href=
"pathmap://RAMSES_GRAPHIC_PROFILES/no.ntnu.item.ramses.graphicprofile.uml#_geG1wIuAEdytf0WtpACIqw"/>
    </eAnnotations>
    <appliedProfile href="pathmap://RAMSES_GRAPHIC_PROFILES/no.ntnu.item.ramses.
      graphicprofile.uml#_UEANMUV1EduH9aQE_pc66w"/>
  </profileApplication>
</uml:Package>
<graphics:Shape xmi:id="_sVLMALymEd6VCq2W8FCNPQ" x="220" y="142"
width="405" height="20" base_Element="_sPgo4LymEd6VCq2W8FCNPQ"/>
<arctis:esm xmi:id="_sd0KobymEd6VCq2W8FCNPQ" base_StateMachine="
_sd0KoLymEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_tgH8MLymEd6VCq2W8FCNPQ" x="222" y="161"
width="404" height="277" base_Element="_sPgo4bymEd6VCq2W8FCNPQ"/>
<arctis:event xmi:id="_4MrlgLymEd6VCq2W8FCNPQ" base_Transition="
_4MYqkLymEd6VCq2W8FCNPQ" triggers="_B2yTULynEd6VCq2W8FCNPQ"
effects="_WCZUMLyrEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_B3-mILynEd6VCq2W8FCNPQ" x="115" y="390"
base_Element="_B2yTULynEd6VCq2W8FCNPQ" orientation="west"/>
<graphics:Shape xmi:id="_DGzJsbynEd6VCq2W8FCNPQ" x="96" y="222"
base_Element="_DFm24LynEd6VCq2W8FCNPQ" orientation="west"/>
<graphics:Shape xmi:id="_ErFm8LynEd6VCq2W8FCNPQ" x="133" y="301"
base_Element="_Ep5UILLynEd6VCq2W8FCNPQ" orientation="west"/>
<graphics:Shape xmi:id="_GLpawbynEd6VCq2W8FCNPQ" x="70" y="262"
base_Element="_GKdH8LynEd6VCq2W8FCNPQ" orientation="west"/>
<graphics:Shape xmi:id="_G-nIwbynEd6VCq2W8FCNPQ" x="130" y="342"
base_Element="_G-Kc0LynEd6VCq2W8FCNPQ" orientation="west"/>
<arctis:event xmi:id="_cACUILLynEd6VCq2W8FCNPQ" base_Transition="
_b_vZMLynEd6VCq2W8FCNPQ" triggers="_B2yTULynEd6VCq2W8FCNPQ"
effects="_WCZUMLyrEd6VCq2W8FCNPQ"/>
<arctis:location xmi:id="_qVWM0LynEd6VCq2W8FCNPQ"
base_Variable="_qVNC4LynEd6VCq2W8FCNPQ" partition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_qVWM0bynEd6VCq2W8FCNPQ" x="367"
y="195" base_Element="_qVNC4LynEd6VCq2W8FCNPQ"/>
<arctis:location xmi:id="_rjwbkbynEd6VCq2W8FCNPQ"
base_Variable="_rjwbkLynEd6VCq2W8FCNPQ" partition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_rjhlkLynEd6VCq2W8FCNPQ" x="370"
y="173" base_Element="_rjwbkLynEd6VCq2W8FCNPQ"/>
<arctis:location xmi:id="_swxGYbynEd6VCq2W8FCNPQ"
base_Variable="_swxGYLynEd6VCq2W8FCNPQ" partition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_sw63YLynEd6VCq2W8FCNPQ" x="247"

```

```

y="174" base_Element="_swxGYLynEd6VCq2W8FCNPQ"/>
<arctis:location xmi:id="_uXSYQLynEd6VCq2W8FCNPQ"
base_Variable="_uXJOULynEd6VCq2W8FCNPQ" partition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_uXSYQbbynEd6VCq2W8FCNPQ" x="248"
y="195" base_Element="_uXJOULynEd6VCq2W8FCNPQ"/>
<arctis:location xmi:id="_v2zqQLynEd6VCq2W8FCNPQ"
base_Variable="_v2zqQLynEd6VCq2W8FCNPQ" partition="_sPgo4bymEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_v2zqQLynEd6VCq2W8FCNPQ" x="514"
y="172" base_Element="_v2zqQLynEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_gJEesLyqEd6VCq2W8FCNPQ" x="231"
y="219" width="149" height="44" base_Element="_gInywLyqEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_hdtWYLyqEd6VCq2W8FCNPQ" x="248"
y="383" width="123" height="44" base_Element="_hdQqcLyqEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_ksip4LyqEd6VCq2W8FCNPQ" x="242"
y="291" width="151" height="44" base_Element="_ksPH4LyqEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_u2FFsLyqEd6VCq2W8FCNPQ" x="249"
y="331" width="108" height="44" base_Element="_u17UsLyqEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_xdt2ULyqEd6VCq2W8FCNPQ" x="236"
y="251" width="157" height="44" base_Element="_xdRKYLyqEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_71L0gLyqEd6VCq2W8FCNPQ" x="409"
y="385" base_Element="_71BdgLyqEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_FbfzYLYrEd6VCq2W8FCNPQ" x="409"
y="219" base_Element="_FbWCYLyrEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_WECS8LyrEd6VCq2W8FCNPQ" x="617"
y="394" base_Element="_WCZUMLyrEd6VCq2W8FCNPQ" orientation="east"/>
<graphics:Shape xmi:id="_XZ3dcbyrEd6VCq2W8FCNPQ" x="616"
y="228" base_Element="_XYrKoLyrEd6VCq2W8FCNPQ" orientation="east"/>
<arctis:event xmi:id="_OhkKkLysEd6VCq2W8FCNPQ"
base_Transition="_OhaZkLysEd6VCq2W8FCNPQ" triggers="
_B2yTULynEd6VCq2W8FCNPQ" effects="_WCZUMLyrEd6VCq2W8FCNPQ"/>
<arctis:event xmi:id="_lHa9ILysEd6VCq2W8FCNPQ"
base_Transition="_lHRzMLysEd6VCq2W8FCNPQ"
triggers="_B2yTULynEd6VCq2W8FCNPQ" effects="_WCZUMLyrEd6VCq2W8FCNPQ"/>
<graphics:Shape xmi:id="_fbM1YUkuEd-L3Z94OHHEpA"
x="621" y="347" base_Element="_famvgEkuEd-L3Z94OHHEpA"/>
<graphics:Shape xmi:id="_kC1e4UkuEd-L3Z94OHHEpA"
x="620" y="316" base_Element="_kCPpAEkuEd-L3Z94OHHEpA"/>
<graphics:Shape xmi:id="_mkqicUkuEd-L3Z94OHHEpA"
x="616" y="282" base_Element="_mj67kEkuEd-L3Z94OHHEpA"/>
</xmi:XMI>

```

Appendix D

Code for Invocation of UPnP Light services

```
import org.cybergarage.upnp.*;
import org.cybergarage.upnp.ssdp.*;
import org.cybergarage.upnp.device.*;
import org.cybergarage.upnp.event.*;
import java.util.ArrayList;
import java.util.Hashtable;

public class UPnPLight extends ControlPoint implements NotifyListener,
EventListener, SearchResponseListener {

private final static String LIGHT_DEVICE_TYPE = "urn:schemas-upnp-org:device:BinaryLight:1";
private final static String LIGHT_SERVICE_TYPE = "urn:schemas-upnp-org:service:SwitchPower:1";

public UPnPLight(){
addNotifyListener(this);
addSearchResponseListener(this);
addEventListener(this);
search();
}

////////////////////////////////////
// Listener
////////////////////////////////////

public void deviceNotifyReceived(SSDPPacket packet) {
String dd=packet.toString();
// System.out.println(dd);
byte[] pp=dd.getBytes();
SSDPPacket ss= new SSDPPacket(pp,pp.length);
// System.out.println(ss.getST());
}

public void deviceSearchResponseReceived(SSDPPacket packet){
String dd=packet.toString();
byte[] pp=dd.getBytes();
SSDPPacket ss= new SSDPPacket(pp,pp.length);
```

```

}

public void eventNotifyReceived(String uuid, long seq, String name, String value) {
System.out.println(""+uuid+"="+seq+"="+name+"="+value);
}

////////////////////////////////////
// Power
////////////////////////////////////

public void Power(String deviceType, String i) {
Device dev = getDevice(deviceType);
if (dev == null)
{
System.out.println("device not found");
return;
}

org.cybergarage.upnp.Service ser=dev.getService(LIGHT_SERVICE_TYPE);

boolean ss=this.subscribe(ser);
if (ss =true) System.out.println(ser + "is true");
Action getPowerAct = dev.getAction("GetStatus");
if (getPowerAct.postControlAction() == false)
{
System.out.println("post nggak ketemu");
return;
}

ArgumentList outArgList = getPowerAct.getOutputArgumentList();
String powerState = outArgList.getArgument(0).getValue();
String newPowerState = (powerState.compareTo("1") == 0) ? "0" : "1";

newPowerState=i;
Action setPowerAct = dev.getAction("SetTarget");
setPowerAct.setArgumentValue("newTargetValue", newPowerState);
setPowerAct.postControlAction();
}

public void SetTarget(String newTargetValue){
Power(LIGHT_DEVICE_TYPE,n);
}

public boolean GetStatus (String deviceType){
Device dev = getDevice(deviceType);

if (dev == null)
{
System.out.println("device not found");
return;
}

org.cybergarage.upnp.Service ser=dev.getService(LIGHT_SERVICE_TYPE);
boolean ss=this.subscribe(ser);
if (ss =true) System.out.println(ser + "is true");

```

```
Action getPowerAct = dev.getAction("GetStatus");

if (getPowerAct.postControlAction() == false)
{
System.out.println("post nggak ketemu");
return;
}

ArgumentList outArgList = getPowerAct.getOutputArgumentList();
String powerState = outArgList.getArgument(0).getValue();
if powerState='1' status=true else false;
return ResultStatus;
}

public int GetLoadLevelStatus (){
//code
}

public int GetMinLevel (){
//code
}

public void SetLoadLevelTarget (int new){
Action setPowerAct = dev.getAction("SetLoadLevelTarget");
setPowerAct.setArgumentValue("newTargetValue", newPowerState);
setPowerAct.postControlAction();
}
}
```


Appendix E

Presenting Embedded Services for End-user Composition

In this case study, the PMG-pro method is applied to support the end-users services composition where run-time services are presented using simple and well-known pictures. After all run-time services are presented, end user can compose the services in a simple way.

1. Introduction

Different perspectives might have different definition of what is a service. From the end-user's perspective, a service can be seen as provided functionality (i.e., at run-time) that can be invoked via their interfaces. With this perspective, a services composition can be seen as a development method of applications. Such development methods (i.e. service-based applications) are currently being adopted in many of today's large-scale software projects [103, 104, 110]. Unfortunately, there is only a few numbers of researches have been done on tools and methods supporting for end-users composition. Having development tools and methods that enable end-user to be involved in the development of service-based application is a challenge.

Using advanced development tools and methods it would be possible for end-users, which typically are inexperienced with technology details, to integrate (compose) run-time services. This case study presents a method of presenting services into abstract graphical representation so that end-users can compose run-time services to build service-based applications. The method is a special case of PMG-pro method that has been presented in [97, 98]. We use service descriptions to generate source code (i.e. for service invocations) and present the source code using abstract graphical representation to represent services

(i.e. service models). The service models are used by the end-users to specify the composition, while the source code is used for the service invocations. For this purpose we use ICE [110], an end-user service composition (i.e., run-time services).

2. A Composition Scenario

In this scenario we also consider that all the devices have been implemented their embedded services using open and standardized languages and technologies. Examples for such open and standardized technologies are Universal Plug and Play (UPnP), Web Services Description Language (WSDL) and Device Profile for Web Services (DPWS). Based on these different embedded services, new applications can be promoted. In this scenario, a new application with the following new functionalities is promoted:

- the application is able to send notifications (e-mail) when the air temperature from the WeatherModule is greater than 50°C,
- it will play music/songs on the Media renderer when the solar radiation is below than 10, and

3. The Development Process

(a) **The Automated Services Presentation Step.**

Since the PMG-pro method is a language-independent method, it is possible to use different existing modeling languages and different modeling editors. This is done by developing and implementing different service abstractors/presenters. The requirement is that the presenter must generate notations (i.e., abstract service models) that conform to the chosen modeling languages. Different representations and notations to represent the existing services can be used. UML classes [80], CORBA components [77], Participants in SoaML [79], or SCA components [40] are among of them. On the system levels, for example SysML [81], a UML profile for system modeling, a building block is used to represent an existing service. It means that in PMG-pro, different modeling languages can be used to model service-based applications. The important thing to remember is that we must take care of the relations (bindings) between graphical representations (i.e. service models) and source code (i.e. implementation for the service invocations).

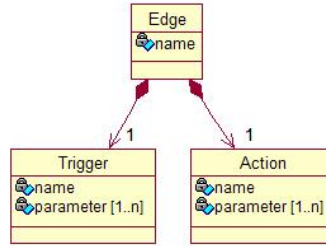


Figure E.1: A simplified UML class diagram of an ICE edge. An ICE edge can be either an action or an trigger edge.

In this case study we will present services using ICE Edges in ICE Puzzle. From a service description (s_k), the abstractor/presenter generates two types of graphical service model. The first type, $M_{s_k}^T$, is a service model type of Trigger, while the second type, $M_{s_k}^A$, is a service model type of Action. The service models conform to a ICE meta-model. The transformation also generates source code (C_{s_k}) that conforms to a selected programming language.

Table E.1: Transformation rules between UPnP - ICE Edge

UPnP	ICE Puzzle
Device name	Egde name
Action	Procedure
Action argument	Signal In
State variable	Signal Out
Type of state variable	Signal type

A conceptual ICE Edge is presented in UML class diagram in Figure E.1. We consider to be able to transform UPnP service desription to ICE Edges using transformation rules presented in Table E.1. Figure E.2 shows a representation of a UPnP service in an ICE Edge as a result of the transformation processes.

(b) The Modeling Step.

Using special modeling editor (i.e., ICE Puzzle), a service-based application can be built by connecting one ICE edge type of Trigger and one ICE edge type of Action. In the composition of ICE Edges, every service model M_s represents either a Trigger $M_{s_k}^T$ or an Action $M_{s_k}^A$.



Figure E.2: An ICE Puzzle represents UPnP MediaRender.



Figure E.3: The models of the new application defined in the scenario. Two puzzle compositions specify the structure and the behavior of the application.

Since a composite application can only be a relation of one Trigger and one Action. Fig. E.3 shows the new application defined in the scenario.

(c) Code Generation Step.

We do not handle the code generation of puzzle composition since it is handled by ICE-core.

4. Discussion

There are two possible service compositions: run-time and design-time service compositions. Several research projects promoting tools and methods for service composition have been conducted, for example the SeCSE project [103]. One of the SeCSE goals is to create methods, tools, and techniques to enable service integrators to develop service-centric applications. The SODIUM [104] project focuses on the need for standards-based integration of heterogeneous services. The two research projects mentioned above were focusing on tools and methods for the development of service-based application. However, the proposed tools and methods are intended for software

developers who know the technology details. Using their developed frameworks, software developers can develop service-based applications and provide them to the end-users. Our work was to provide a method for end-users (i.e., inexperience of technologies) are able to compose run-time services to promote new applications.

The fact that different perspectives may have different definitions of a service, the definition of a service composition may also be different. We use PMG-pro methods to present services so that end-users can compose (integrate) run-time services. Using the PMG-pro method it would be possible to generate service graphical representation that can be used by end-users (i.e., run-time composition). We present a service using a simple and well-known picture. Each service graphical representation has source code implementing the service invocations. The code for each service graphical representation is packed as a bundle (i.e., ICE edge). The ICE Edges then can be used by end-users to model the composition (at run-time) while the source code is used by ICE to execute the composition.

5. Summary of the Use Case

In this case study we have shown a method for presenting run-time services for supporting for end-users composition. To show the idea, we use PMG-pro, a language-independent, bottom-up and model-driven method. Based on existing service frameworks and service descriptions, a service presenter/abstractor captures statically platform knowledge and presents them to then users using abstract graphical representation (i.e., ICE Edges). Using simple and well-recognized pictures to present run-time services it would be possible for inexperience users to build service-based applications using run-time services.

Index

- Abstraction, 9, 102, 133
- Abstractor, 9, 61, 134
- Acceleo, 50
- Activity diagram, 46
- Apache, 38
- API, 36, 43, 60, 72, 117, 121, 132
- Architecture, 29, 33, 36, 41, 44, 127
- ATL, 50
- Automated, 9, 10, 134
- Automatic, 53
- Axis, 38, 60

- Building block, 67, 103

- CC/PP, 56
- Choreography, 126
- Cloud computing, 3, 6, 14
- Code generator, 57
- Component , 33, 127, 132
- Concrete service, 16
- Control point, 54
- Cyberlink , 36, 60, 88

- DPWS , 21, 40, 41, 60, 63, 71, 134
- DPWS framework, 41
- DSML, 22, 47

- Embedded, 23
- Embedded system, 23
- End-users, 134
- Event, 40, 98–101, 103, 116
- Event-based, 98, 100, 103, 116
- Event-based system, 98

- Framework, 40, 41, 43, 72
- Future Internet, 1–3, 6, 120

- inexperienced users, 3
- Internet of object, 2
- Internet of services, 8, 24, 38, 133
- Internet of Things, 2–4, 8, 23, 52, 56, 57, 105, 113, 131

- JET, 50
- Jini, 30, 43, 44, 53, 63

- Library, 71, 72

- M2M, 49, 50, 72
- Model transformation, 49, 118
- Model-driven, 8–10, 24, 93
- Model-driven development, 8, 24

- Notation, 22, 50

- Object, 125
- Ontology, 74, 132
- OSGi framework, 42, 43, 97
- OSGi services, 21, 63, 97, 105, 134

- Parser, 87
- Participant, 35, 64, 67, 118
- PLA, 55
- Presentation, 10, 102, 107, 129, 134
- Providing, 134

- Rational Rose, 84, 92, 94, 96, 98, 115, 125, 132, 133
- SBIS, 13–15, 52
- SCA, 33, 34, 64, 67, 118, 127
- SCV, 55
- SEIS, 13, 14
- Service creation, 5
- Service model, 125
- Smart home, 73, 93, 94
- SOA, 4, 15, 21, 29, 30, 36–38, 40–43
- SoaML , 34, 35, 67, 118
- Software developer, 10, 66
- Software product line, 55
- Software unit, 32
- Solution, 16
- SPL, 55, 56, 114
- SSDP, 101
- T2M, 49, 50
- T2T, 49, 50
- Text transformation, 118
- Unified modeling language, 21
- UPnP devices, 16, 21, 36, 66, 67, 98–100
- UPnP framework, 36
- UPnP services, 16, 21, 36, 38, 63, 84, 86, 88, 89, 92, 94, 97–100, 102, 107, 110, 116, 134
- Web services, 16, 21, 22, 33, 37, 38, 41, 42, 61, 63, 105, 126, 127, 134
- XML UPnP service description, 67, 72, 87, 118
- XML-based service description, 63, 98, 107, 115