Methodology for Service Development in a Distributed Smart Home Environment

Master Thesis

by

Cezary Szczegielniak

presented at

Department of Software Systems
Hamburg – Harburg University of Technology
Germany

October 2006

Tutors:

Prof. Dr. Joachim W. Schmidt,
Prof. Dr. Dieter Gollmann,
Hamburg University of Technology

Dr. Asa MacWilliams, Siemens Corporate Technology

Declaration

Hereby I declare that I am the author of this thesis, titled “Methodology for Service Development in a distributed Smart Home Environment”. All literally or content related quotations form other sources are clearly pointed out and no other sources rather than the ones declared are used.

Munich, October 10, 2006

Cezary Szczegielniak

Acknowledgment

I would like to thank Professor Schmidt and Rainer Marrone of Software Systems Group for supervising this thesis and giving feedback for continuously enhancing the scope of this work. Thanks also to Professor D. Gollmann for being the co-supervisor of this thesis. Furthermore, I am very grateful to Dr. Asa MacWilliams for giving constructive feedback and supervising the work at Siemens, Corporate Technology in Munich. Furthermore I would like to thanks to whole Smart Home team for giving me valuable suggestions and hints during this work.

Abstract

Smart home environments are aimed at enabling the integrated remote control of various areas of a house such as lights, security, and entertainment or home appliances. The smart home architecture consists of various components which can be built in different technologies. In particular, in the Siemens Smart Home platform (currently under development), there are three different extensible plug-in platforms: OSGi, .NET and EJB.

A user benefits from smart home environments via various services which are developed by an operator or a 3rd party. In the smart home architecture one deals with distributed services, since an implementation of a service can reside on different components. In the worst case, a service is distributed among various, technologically distinct platforms, which have different graphical locations. As a result, the development process is especially difficult.

This thesis introduces a method to develop smart home Services. To achieve this goal, the Siemens Smart Home architecture and associated technologies are analyzed. Based on such studies a tool chain, to support service development in Siemens Smart Home environment, is defined. The tool automates common development activities, which include: user interface design (device specific), specification of application logic, composition of existing services, configuration to end-user needs, debugging, event reporting, deployment, etc.

Having a defined set of tools, the current software market is investigated. If any tool already existing on the market fits the need of the chain tool, it should be included in the set. The remaining components, which are not available, have to be developed.

The problem of developing services for the Siemens Smart Home can be perceived as a general problem of developing services in a distributed, heterogeneous environment. This solution in this thesis is consulted against similar projects, for example other smart home project, automotive projects.

The practical part of this thesis is to choose and develop one or two software components from the previously defined tool chain.

The evaluation process gives the information about the quality of the tool chain and implemented components. The feedback will be gathered from service or smart home developers.

Contents

List of Figures

List of Abbreviations

.NET CF	.NET Compact Framework
API	Application Program Interface
BPEL	Business Process Execution Language
BSS	Business Support System
CLR	Common Language Runtime
CRC	Class-Responsibility-Collaboration
CVS	Concurrent Version System
DCP	Device Control Protocol
DECT	Digital Enhanced Cordless Telecommunication
EIB	European Installation Bus
EJB	Enterprise Java Beans
FGAG	Functional Group Addressing Guidelines
FXL	Flexible XML-based Languages
GENA	General Event Notification Architecture
GST	Global System for Telematics
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JDT	Java Development Tools
JNDI	Java Naming and Directory Interface
IP	Internet Protocol
JTS	Java Transaction Service
JVM	Java Virtual Machine
LAN	Local Area Network
MIT	Massachusetts Institute of Technology
NAT	Network Address Translation
OSGi	Open Services Gateway Initiative
OSS	Operation Support System
PC	Personal Computer
PDA	Personal Digital Assistant
R&D	Research and Development
RC	Remote Control
RCS	Revision Control System
SDK	Software Development Kit
SDM	Service Deployment Manager
SDSS	Service Development Support System
SEP	Service Enabling Platform
SGP	Service Gateway Platform
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Service Language Layer
STB	Set-Top-Box

SWT	Standard Widget Toolkit
TCP	Transmission Control Protocol
TL	Transformation Language
UI	User Interface
UIEP	User Interface Extensibility Platform
UML	Unified Modelling Language
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
VoIP	Voice over Internet Protocol
WAP	Wireless Application Protocol
WLAN	Wireless Local Area Network
WML	Wireless Markup Language
WS	Web Servicess
WSDL	Web Service Definition Language
XL	XML Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

1 Introduction

A smart home is the term for a domestic environment equipped with modern technologies. The various technologies are integrated, allowing devices using different communication protocol to speak to each other. This aims to improve quality of life, security and communication with the external world. The picture below presents an example of a smart home. The house was built in Berlin for test and marketing purposes [54].

Figure 1.1: T-Com-Haus – device control [65]

The picture was made in the T-com-haus in Berlin. The woman at the picture can control home functions (e.g. dim lights) via a PDA.

The architecture of the smart home consists of various distributed platforms (smart home devices, gateways and application servers). Currently there is a prototype of the Siemens Smart Home solution. Some architecture parts of the platform are still under development. For this reason, best practices of service development for this highly distributed and heterogeneous environment have not been fully investigated. A good methodology and a tool chain which can simplify this process are desperately needed.

In this introduction the terms smart home and more general heterogeneous environment are introduced. Next, the addressed problem, proposed solution, approach and contribution are briefly described.

1.1 Smart Home

In the Siemens Smart Home various technologies and stand alone solutions (e.g. security, light control, entertainment) are integrated and can cooperate with each other. This integration improves the quality of life and security to a larger extent than stand alone solutions.

Siemens Smart Home architecture consists of several basic components: backend server, home gateway, controlled devices and control devices. Due to different requirements for a given component and different device capacities, the platforms deployed in smart home architecture are de-

signed to use different technologies. In particular, in the Siemens Smart Home platform, there are three different extendable plug-in platforms based on the following platform technologies: OSGi (home gateway), .NET CF (PDA's) and EJB (backend server).

A user benefits from a smart home using various services deployed on the above listed components. This requires a deep understanding about smart home architecture and different platforms used there, since the application logic can reside on each of the components.

Due to the use of different technologies, the smart home can be considered as a special case of a heterogeneous, distributed environment. It makes the service development to a large degree similar in both cases. Therefore, the studies presented in this work are helpful in other domains which suffer similar problems (e.g. automotive, aviation industry).

1.2 Addressed Problem

In smart home, services are distributed over various, heterogeneous components. In order to implement such a service, a developer must have a broad knowledge about the domain and all used technologies. This process is time consuming already before the real development starts. The next problem arises when an engineer, during service development, has to take many design decisions (e.g. service architecture, implemented components, etc). For anybody inexperienced in smart home domain this may lead to wrong decisions and delays in later phases of the project.

Also other development tasks (e.g. implementation, testing, etc.) might be speeded up by increasing the automation level (e.g. use of generators).

1.3 Proposed Solution

To address these problems, a method for service development in the smart home domain is proposed. The key element of this solution is a tool chain, which speeds up the development process and makes it less error prone. Some of the tools in the set, such as the service architecture chooser, help the programmer to make a good design decision without the need of deep investigation of the smart home domain. Other tools, such as various generators, already generate much of the code, leaving the developer to simply implement service specific logic. In this case the programmer does not need to concentrate himself to a large extend, with technology specific code and integration issues.

1.4 Approach

This master thesis woks out a method of developing services in the Siemens Smart Home environment. The present development process is analysed and various tools which aims at improving the process are proposed. The method is also related to the general problem of developing services in distributed, heterogeneous environments. The structure of this work is as follows.

Chapter 2 is an introduction to the smart home domain. The smart home definition, services, and some R&D projects are presented. Additionally some projects in other distributed, heterogeneous domains are introduced.

Chapter 3 presents the Siemens Smart Home, its architecture and underlying technologies, i.e. OSGi, EJB, .NET CF, UPnP. The analysis of Siemens Smart Home services is carried out.

Chapter 4 gives the overview of general software development and service development process. Differences and similarities between these two processes are discussed.

Chapter 5 lists and describes various tools which are available on the market, for service development. The applications are classified into standard tool (for general software development) and specific for the smart home domain. At the end the FXL framework is introduced and its relevance to the smart home domain is discussed.

Chapter 6 presents the current service development process for the Siemens Smart Home. After that the process is extended with a number of tools whose aim is simplify and speed up the service implementation activity. Next, the test activity together with various test applications is described in more detail. Further, a sophisticated Service Development Support System (SDSS) is introduced.

Chapter 7 presents an example tool implementation: the Siemens Smart Home scervice wizard for Eclipse platform. Its architecture, XSLT approach, and a Virtual Devices generator are described.

Chapter 8 describes the use of the worked out method for the development of two concrete services Device Control and Scene Control service. During the process the implemented Eclipse wizard is used.

Chapter 9 verifies the results of this work. The evaluation is based on the interviews with people involved in Siemens Smart Home platform development.

Chapter 10 summarises the contribution of this work, its limitations and possible extension.

1.5 Contribution

Firstly, this thesis works out the development method for Siemens Smart Home services. The development process, illustrated with various flow diagrams, can be constituted as guidelines for service programmers.

Moreover the process incorporates a number of supporting software tools. Some of these applications, already available on the market, are listed and shortly described in the thesis. Furthermore, many new tools for the special purpose of the development process are designed.

Prototypes of the three proposed tools are implemented: *Service Architecture Construction tool*, *Virtual Devices generator* and *OSGi Service bundle generator*. These applications are included in the Eclipse plug-in: *Siemens Smart Home Service wizard*.

Beyond Siemens Smart Home, general concepts of service development method presented in this thesis are applicable to other distributed, heterogeneous environments.

2 Smart Home

This chapter introduces a reader to the smart home domain as a highly distributed, heterogeneous system. Next, underlying technologies and development challenges are presented. At the end various projects from the smart home domain and from other domains, as examples of heterogeneous environments, are described.

2.1 Definition

A good definition of a smart home comes from the TU Berlin:

> *"… A Smart Home is the integration of technologies and services in the domestic environment in order to improve quality of life, security and communication possibilities with the external world."* [16]

According to this definition, all different technologies which are currently used in the domestic environment, must cooperate with each other (see figure 2.1). The integration is a crucial factor that improves the control and quality of life at home. It is possible to control, from just one control device, all the appliances. A classical example can be setting mood to romantic in a mood control service. As a consequence, all the lights in a room are dimmed, blinds are lowered, a suitable music is played and a TV set displays appropriate scenery. Without an integrated smart home solution, all those actions would have to be done manually.
The ability of the system to communicate with the external world gives the possibility of offering more sophisticated services (e.g. health care services, up-to-date cooking books).

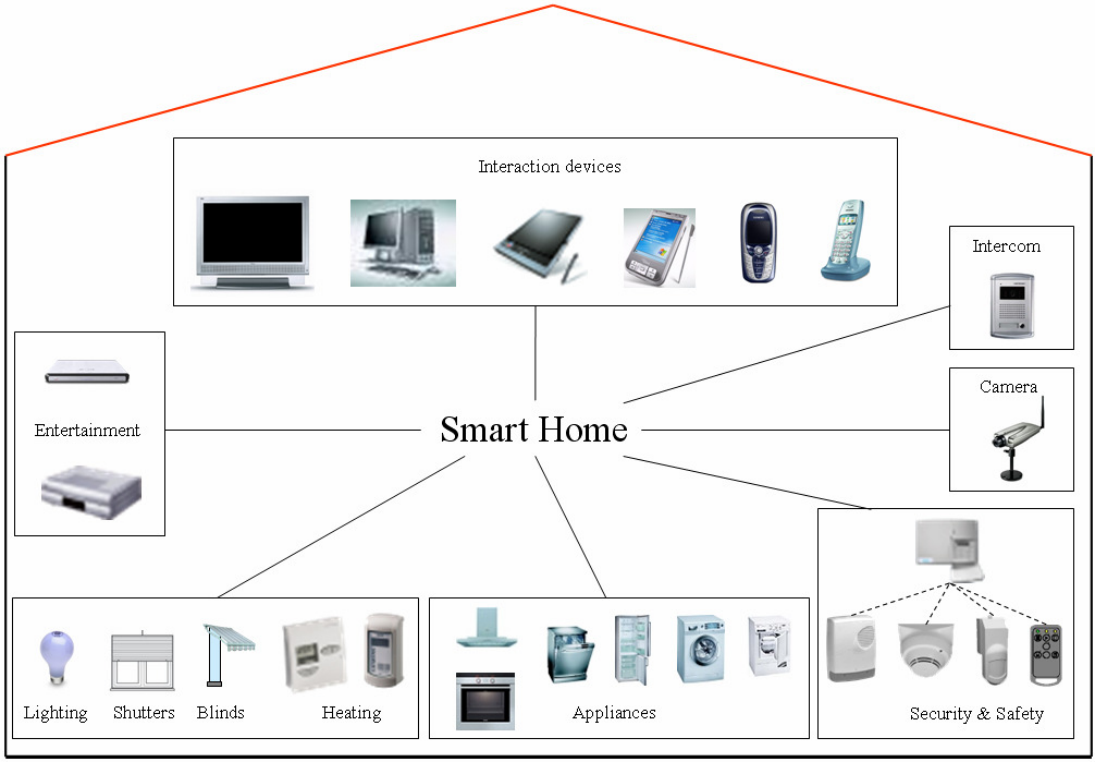


Figure 2.1: Integration of various single-craft solutions in a smart home [64]
The picture presents various single craft solutions integrated in a smart home.

2.2 Distributed, heterogeneous environment

A smart home is an example of a distributed, heterogeneous environment. Various single-craft solutions exist at home and are often distributed in the whole house (electric installation automation, security). These solutions often use different standards such as: communication protocols, software and hardware technologies.

2.2.1 Various technologies at home

Currently in the smart home market there are many different single-craft solutions, addressing one single area of home automation: household appliances, entertainment and communication solutions. In the field of automated electrical installation there are many standards: LonWorks [29], X10 [68], KNX [12]. All the technologies have some actuators and sensors scattered over the whole house which communicate over a proprietary protocol. The whole network of these devices enables inhabitants to control lights, blinds, heating and air conditioning remotely.

Another isolated solution is a home security system. As in the previous example, there are many sensors and actuators located in different places in the home. They communicate, usually over a proprietary protocol, with a central unit. The central unit is able to control all the devices (arming, disarming) and to alarm the right duties (in case of burglary or fire alarm).

2.2.2 Integration needs

As mentioned above, all the smart home solutions add significant value to a residential house. However, like in the Siemens Smart Home, the benefit is much bigger when all the technologies are able to speak to each other. This enables the deployment of more sophisticated services and thus provides better quality of life to inhabitants.

A good example is a mood control service as was already described above. Without the integration of technologies, the implementation of it would be practically impossible.

Another interesting single-craft solution for a family house is a doorbell service. Most of the current products allow the inhabitants to see, via a video camera, a person wanting to enter the building. This can avoid allowing unwanted people to enter the house. One thing that might be uncomfortable here is the necessity to go to the speaker device located somewhere near the front door. Moreover, in a big house, it might even be hard to hear the ringing bell. In an integrated smart home it is easily possible to redirect the incoming video conversation to the other devices in the home: TV sets, Set-Top-Boxes, PDAs, cord phones. The decision, to which device to make the redirection, can be based on the presence sensors located in the whole house. In this case the presence sensors belong to a security solution which uses another technology than a doorbell. When there is nobody at home the doorbell, with the help of a gateway to the external world, can redirect the conversation to a mobile phone or a PDA outside the house. When an inhabitant recognizes somebody as a close friend, he or she can let him in.

2.2.3 General smart home architecture

The definition and cooperation described above between various solutions in the home impose a general smart home architecture. Since the different technologies are basically not able to speak to each other, the integration can be done via current standards such as Web Services [63] or UPnP [56]. These standards can be supported directly by a single-craft solution, or by implementing special bridge software on a central computation module unit (gateway). The task of the bridge is to

make transformations between protocols (proprietary and integration protocol). The gateway element makes the implementation of more sophisticated services easier. These services, like mode management, must have access to all single-craft solutions at home. The gateway is the perfect element for the deployment of such applications, since it has access to all home devices.

According to the smart home definition, there must be a connection with the outside world. In enables the use of internet and other carrier services, as well more advanced services offered by 3rd parties (Video on Demand, weather information, etc.).
The central unit – gateway, is also the best module for integration with the control devices. These devices, such as PDA's, phones, PC's must be able to have access to different services running on the gateway. The generic architecture is depicted on the figure 2.2.

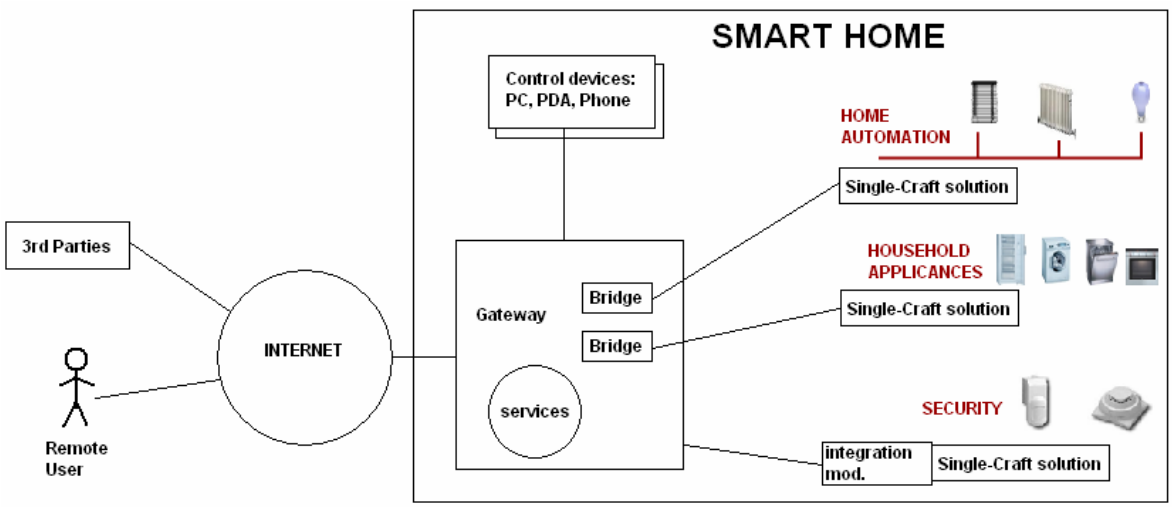


Figure 2.2: General smart home architecture

The picture presents general smart home architecture. The gateway is a central element at home. All single-craft solutions as well as control devices (PC, PDA …) are connected to the gateway. The gateway has access to internet, which makes it reachable by 3rd parties and remote users.

2.3 Smart Home services

A user benefits from a smart home via a number of applications deployed on various hardware architecture components. A single application is a piece of software which consists of UI interface, service logic. While the UI interface is deployed on control devices (PDA, PC, etc.), the service logic might be distributed over various components: gateway, control devices, controlled devices, 3rd party servers outside the home network.

2.3.1 Examples

A long list of different smart home services is provided in J.Wechsler [64] master thesis. Some of the interesting examples are:

Video on Demand
The video is streamed via the Internet to the Receiver. Streaming means that the picture is displayed while a transmission is still in progress.

Follow-Me Media Streaming

The media is streamed to the device at home, based on the current location of an inhabitant. When a user moves around the house, media is played on different devices following the inhabitant's location.

Traffic Information
The personalized traffic information is provided via the Internet to the user. The notifications warn the user when abnormal situation arises, e.g. a traffic jam due to an accident on the road to work.

Video Telephony
This is an enhanced standard telephony, so a user can see the person he/she is speaking with.

Vital Signs Monitoring
The vital signs of a user are constantly monitored via a special device. If the measured value enters a dangerous zone (e.g. a blood pressure too high), an alarm is raised and relevant authorities are notified.

Tele Medicine
Using this service, a doctor might be consulted from home via the Video Telephony.

Presence Simulator
This is a kind of a security service. When nobody is at home the devices (lights, blinds, TV sets, etc.) can simulate the presence of inhabitants.

Energy Saving
Energy might be saved by turning off some devices when they are not needed, e.g. turning of lights in a room when nobody is there. Further the service can schedule some energy intensive tasks (washing, drying, etc…) for the time when the energy costs are low (e.g. at night).

Mood control (Scene Control)
This service, already described, enables a user to activate a previously programmed set of actions.

2.3.2 Development challenges

A smart home service application is distributed over various components (see figure 2.3), e.g. UI Interface on the PDA, service logic on the gateway or 3rd party server. These distributed component usually use different core technologies. For example, it is very likely that a 3rd party server will be based on EJB technology [52], while .NET Compact Framework [67] is a common platform on PDAs. In fact each part of a service is implemented using various technologies. These parts need in the end to communicate seamlessly with each other and with deployed single-craft solutions.

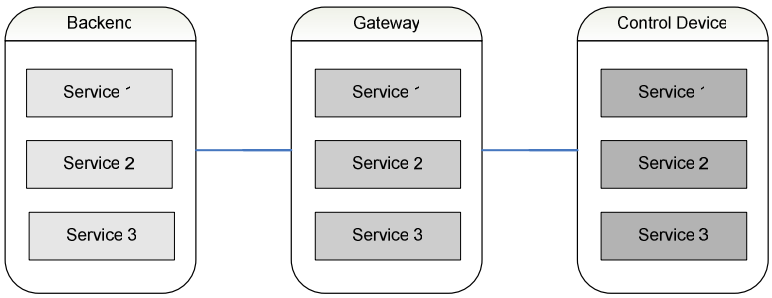


Figure 2.3: Services distributed over smart home components
The picture presents three smart home components (frames with round corners): Backend, Gateway and a Control Device. A single smart home service is distributed over these components, e.g. Service 1 consists of three parts what is symbolized by rectangles in different grey shades.

As a result, a developer must have a broad knowledge about all used technologies and possibly complicated service deployment. The next problem is testing, which is not an easy task in such a distributed, heterogeneous environment. In this case, when a bug appears, it is not easy to locate the source of the error. Moreover a single service can possibly make use of many other single-craft solutions (security, lights automation, etc.). In order to test such a service, all these technologies are needed. Having a well equipped smart home lab often means a large investment. While the market for intelligent home solutions in not yet a mass market, this investment can be a problem.

2.4 Smart Home R&D Projects

There are not many integrated smart home solutions on the market. The ones available are usually made to individual order, making them very expensive. However, if smart home becomes more popular, the costs will drop down and the market will grow significantly. This can be clearly seen in mobile phones market where prices have dropped. Therefore, there considerably are many smart home R&D projects from various commercial and non commercial institutions. Some of the example are listed and briefly presented below.

2.4.1 LIVEfutura

The LIVEfutura project is conducted at Fraunhofer Institute in collaboration with many commercial companies [26]. The institute tries to make one, comprehensive system which enables the control of the whole living environment (house, car, summerhouse, etc.). There are several main aims of the project:

- System integration of heterogeneous technologies via gateways and middleware,
- Place and control device identification – in order to provide more adequate user interface
- Adoption of IP-functions to support IP-based local and remote communication,
- Implementation of integrated service platform
- Realization of user-friendly, multimodal service interfaces

For the system integration and service platform an OSGi [34] platform was used. In this case the benefits of the service oriented platform (OSGi) are:

- Dynamic deployment of innovative services based on user preferences,
- Interconnection of diverse home devices,
- Remote control, maintenance and diagnostics,
- Communication between car and home,
- Support for multiple visual interfaces and control points for home system.

The OSGi platform will be described in more detail later in this work.

The research focuses also on the user side and tries to answer the question; “What would people find useful for the home?” User experiences with a smart home are not really known. Finding out what kinds of services really have value for inhabitants is a basic factor for further market success.

2.4.2 Intelligent Room

The research lab, called Intelligent Room, for exploring the design of intelligent environments was created by MIT [7]. The goal of the project is to “*allow computers to participate in activities that have never previously involved computation and to allow people to interact with computational systems the way they would with other people: via gesture, voice, movement, and context*”. For this purpose many non-visible interaction devices, such as microphones, cameras, sophisticated sensors, are used.

For the technology controlling purposes, a modular system of software agents (*scatterbrain*) is adopted. The system consists of 30 distinct, intercommunicating software agents running on ten different workstations. Similar to OSGi platform in the previous example, the *scatterbrain* is an element integrating all the technologies used in the Intelligent Room.

2.4.3 EasyLiving

EasyLiving project from Microsoft Research aims at development of architecture and technologies for intelligent environments [9] [5]. It concentrates mainly on three aspects:

1. UI and application,
2. Sensing and World modelling,
3. Distributed System.

The first aspect, UI and application, concentrates on development UI on different interaction devices, like PC, PDA, mobile phone, etc. Additionally different applications, creating the functionality of a smart home, are developed.

Another important developed aspect is world modelling and sensing. Many efforts in the project are made to make a system which is able to adopt itself to the current state of domestic environment. The aim of this is to improve quality of life, performing many user actions automatically or with minimal interaction. To perform this task the system must learn form typical behaviours of inhabitants. The adaptation process can be successful when, on the one hand a good model of the domestic environment is provided, and on the other hand sensing abilities are good enough. For example, when a user goes to the bathroom at night the system will recognize this activity and automatically switch on light.

The last research is preformed in the distributed system field. In order for the whole system to work properly, all its parts must communicate with each other. This means that many heterogeneous technologies must be integrated. For this purpose, a middleware solution, InConcert, was developed.

2.4.4 Siemens Smart Home

The main goal of the Siemens Smart Home project is to integrate various single-craft family home solutions. This is achieved with the help of OSGi platform and UPnP technology. The system is based on Service oriented architecture, which enables the deployment of services according to the user preferences. More detailed information about the Siemens Smart Home project can be seen in the next chapter (3 Siemens Smart Home).

2.5 Projects in other domains

The attempts to improve comfort quality of live are not only in domestic environments. The entertainment systems are strongly investigated in such domains as automotive and aviation industry. Architectures in these domains are often similar to the one used in the smart home environment (e.g. different platforms). Therefore, the research presented in this paper might be useful also in these areas. Some projects from other domains and brief description of them are presented below.

2.5.1 Automotive

In recent years the use of software components in cars has grown exponentially and become a significant factor in the automotive industry [3]. Nowadays, almost every new innovative feature in a car is supported by software. This is especially true in entertainment and information platforms, which are becoming more and more sophisticated (e.g. DVD players, navigation system). The architectures of these systems are often similar to many smart home projects, e.g. they use OSGi [34] and SOA as an integration platform (see figure 2.4). In cars, as at home, a gateway platform is deployed. It is connected to all car devices and to an external backend system. Therefore the problem of developing a service is similar for automotive and home domains.

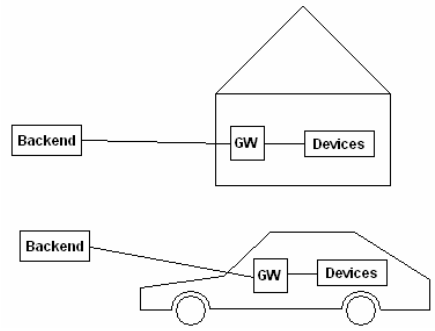


Figure 2.4: Smart home vs. automotive systems
The picture presents an overview of smart home (upper) and car (lower) architectures. For both architectures, the three components (Backend, GW – Gateway, Devices) are similar.

GST
The GST project (Global System for Telematics) is sponsored by the European Commission [15]. It tries to create an environment in which innovative telematics services can be developed and delivered in cost effective ways. For this purpose a GST Service Platform specifies a set of services and library components which enables the:

- Capturing of vehicle data (e.g. speed, location),
- Communication with remote service and GST control centre,
- Visualization of end-user information.

A GTS-compliant Application Runtime Environment can run on any adapted framework such as J2ME/OSGi or .NET Compact Framework.

By defining "building blocks" in automotive software, standard telematics functionality can be brought to all GST-compatible cars and on-line services can be delivered. This move towards an open system enables services to be de developed by third parties via a defined API. The services can than be downloaded by a user, depending on preferences. This service model is quite similar to a smart home architecture based on SOA.

Autosar
Autosar project is a partnership project between many car manufactures and component delivery companies [2]. Its goal is to specify and standardize architecture elements. The Hardware independence will be supported by standardized interfaces and their communication mechanism (decoupling hardware from software). This enables the realization of Standard Library Functions. Like in smart home projects, in Autosar one deals with many different, heterogeneous technologies. Having a platform integrating all the components, the development of new, more sophisticated ap-

plications should be possible. Having standardized interfaces these applications could be developed by 3rd parties.

2.5.2 Aviation

In the aviation domain, similar to the automotive domain, the need for software and electronic components has grown significantly in recent years. More than 30% of civil aircraft development costs are devoted to system and electronic equipment. The cause of the increased magnitude and complexity of electronic functions is, on the one hand, new requirements form pilots, airlines, aviation authorities, and on the other hand, the need for safe, efficient, cost effective and user-friendly aircraft.

Victoria
The Victoria project [58] aims at addressing these problems. The project brings together 34 industrial companies, R&D centres and universities from the Europe Aerospace community. Victoria tries to develop a new aircraft electronic system based on IME (Integrated Modular Electronics) technology and architectures. Via the Open System concept and standardizing a set of interfaces between basic resources and applications should open up the market.

Part of the project is dedicated to in-flight entertainment systems. These systems evolve rapidly attempting to offer a passenger all the entertainment services he normally finds at home, e.g. includes Video on Demand, Music on Demand, Internet, telephony, e-mail. These services require an architecture consisting of a backend system (on the plane) for managing user terminals and storing large amount of data (Video on Demand). The terminals, which are gateways and UI devices at the same time, offer passengers various services. Therefore, like in a smart home, a service has to be developed for various distributed platforms.

3 Siemens Smart Home

In this chapter the Siemens Smart Home solution is presented. The architecture and its individual parts are described in more detail. Further, a brief overview of some used technologies (such as UPnP, EJB, .NET CF and OSGi) is given. At the end a service and service classes, in the context of smart home, are introduced.

3.1 Introduction

Siemens Smart Home is an R&D project which aims to improve quality of life, security and communication possibilities in a domestic environment [48]. This is done mainly by integrating various single-craft solutions available on the market like security, entertainment, white goods, etc. A user can control all the devices from the single control point, e.g. PDA, PC or DECT phone. The architecture allows also to offer more sophisticated services, e.g. tele medicine, presence simulator and scene control. Another basic idea of the Siemens Smart Home is the seamless integration of services into the system at home upon a user request.

The platform is mainly targeted at telecom operators and service providers. In the recent rapid growth of the Internet, use of VoIP communicators and popularity of mobile operators, the revenue from classic telecom services is drastically decreasing. Therefore, the companies from the “old IT” industry are strongly looking for new ideas and business opportunities which enable them to use existing network infrastructures.

3.2 Architecture

The key concepts of Siemens Smart Home architecture are the ease of integration of various single craft solutions from different manufactures and the possibility to deploy 3rd party applications. The modular and open standard architecture follows SOA paradigm [46]. In this way, a user benefits from a smart home from various services deployed in the system.

A Service Gateway Platform (SGP) is a central software element at home (see figure 3.1). It communicates with controlled devices from one side (e.g. dimmers, switches, security sensors, surveillance cameras), and with control devices from the other side (e.g. PC, PDA, DECT phone). An OSGi framework [34] runs on the gateway, which provides an easy way to deploy smart home services as OSGi bundles. A set of User Interface Extensibility Platforms (UEIP) enables runtime extensions for new services and offers core services for connectivity, subscribing, etc. The communication with the outside world is made via the SGP connected with backend server (SEP) located outside home via a management link. The Service Enabling and Management Platform (SEP) allows to control the gateway in convenient way, deploy remotely new services upon user requests and offer more sophisticated services which require a powerful backend system (e.g. video on demand).

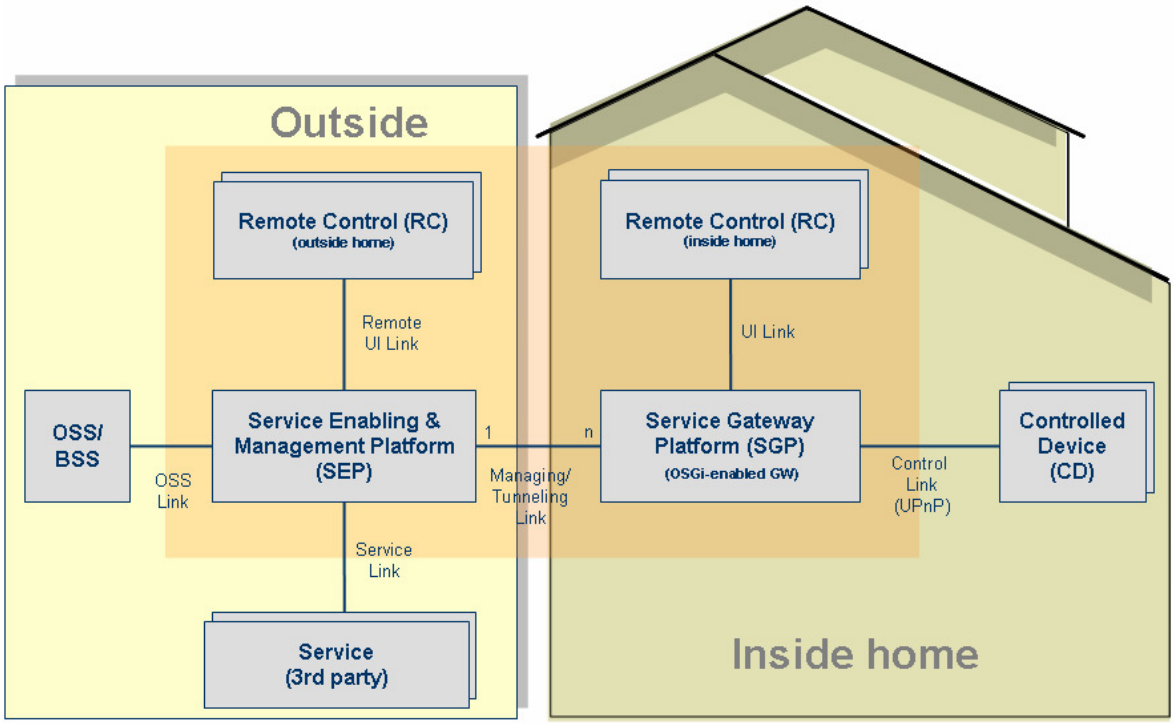


Figure 3.1: General Siemens Smart Home architecture [35]

The picture presents Siemens Smart Home architecture. Rectangles present various architecture components. They are located inside the home (e.g. SGP) as well as outside the home (e.g. SEP).

3.2.1 Controlled Devices

All controlled devices (e.g. security devices, electrical installation, white goods appliances) should offer a control interface, send events and current status information to the Service Gateway Platform (SGP). For example, Siemens home automation solution in electrical installation field [14] is based on European Installation Bus (EIB) standard [24]. Actuators (e.g. dimmers, switches) can send events when somebody switches on or dims a light. Additionally inhabitants are able to see the current status and control an EIB device from a remote control panel. This feature applies to all single-craft solutions used in the Siemens Smart Home (e.g. white goods [43]).

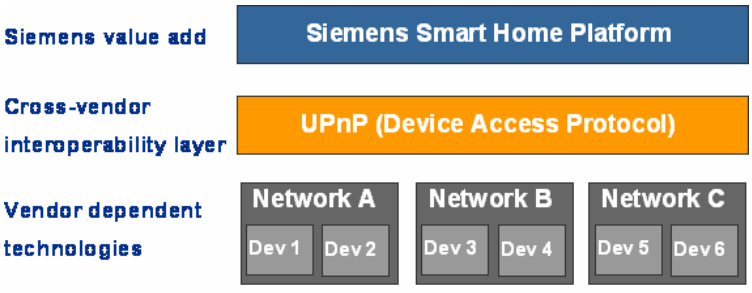


Figure 3.2: UPnP integration layer [35]

The picture presents how various home devices are integrated in the Siemens Smart Home. On the bottom layer are various devices (vendor dependent technologies). They are integrated using UPnP protocol (second layer). At the top is the Siemens Smart Home Platform.

Various single-craft solutions are connected to the SGP via the IP protocol. Therefore devices must be accessible via the IP protocol, either directly or via a bridge. Since the control itself is performed via the UPnP protocol (refer to chapter 3.3.4), devices must support the UPnP. The UPnP protocol is an integration layer for different third party single-craft solutions (see figure 3.2).

The ideal solution is when devices support UPnP protocol out of the box. In this case they can be connected in a plug-and-play way to the network via Ethernet or WLAN. The system running on the SGP automatically discovers the devices and makes them available via Siemens Smart Home API functions (see figure 3.3: A and B cases). The situation is more complicated when the devices are connected via a non-IP network and do not support the UPnP protocol. In this situation additional bridge software must be provided. The bridge makes an IP to Non-IP protocol conversion and translation between UPnP to a proprietary control protocol (e.g. translation from EIB protocol to UPnP).

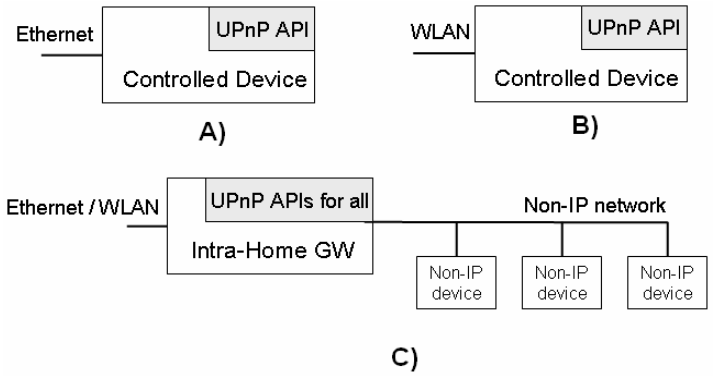


Figure 3.3: Various connection possibilities of controlled devices: A) via Ethernet and devices supporting UPnP protocol; B) via WLAN and devices supporting UPnP protocol; C) via Ethernet/WLAN and the UPnP bridge [35]

3.2.2 Control Devices

Inhabitants can control home appliances via various remote control devices. On the devices two types of UI clients are supported: Rich and Web browser clients. The first gives better UI experience and requires deployment of special software on a rich client device. The second is less powerful in terms of the GUI, but the control device needs only a browser installed. Currently in the Siemens Smart Home Web browser clients are supported by PDAs, PCs, Laptops, mobile phones (WAP), STBs, and rich clients by PDAs.

The control devices are connected with SGP via a UI Link and with the back end system via a Remote UI Link (when devices are accessed from outside the home). The exact communication protocol is specific for each remote control device category and is handled by a protocol adapter plug-in in the Service Gateway Platform or in the Service Enabling and Management Platform.

One of the basic ideas of the Siemens Smart Home is the seamless integration of services into the system at home upon a user request. This includes deploying various software parts on a number of hardware components. For the flexible updates of the rich client on a PDA a UI Extensibility Platform (UIEP) in .NET Compact Framework was developed. Thus the platform running on a PDA can be updated by a new UI functionality at runtime. When a developer writes a UI plug-in component for .NET CF, the plug-in is automatically downloaded and installed on the PDA. Furthermore, the UIEP platform offers a number of core services for connectivity, subscribing, logging, service access, etc.

3.2.3 Service Gateway Platform

The Service Gateway Platform (SGP) is a central component and runs on a residential Service Gateway. The SGP is connected via the IP protocol to:

- Controlled devices, e.g. EIB devices, Security devices,
- Control devices, e.g. PDAs, PCs,
- Outside world and Service Enabling Platform (SEP)

The SGP is a service execution framework with such features as: life-cycle management, dynamic updates, and support for multiple UI interfaces, authentication and access control. The gateway are to provide access for inhabitants to subscribed services and controlled devices via remote controls, and to provide access for subscribed services to controlled devices. The platform follows also the plug & play concept for new devices and services. This means that new devices connected to the home network are automatically discovered (via the UPnP), the corresponding services are downloaded and installed. UI architecture is also updated so the user interaction is brought to a minimum. Upon the subscription to new services, the needed components are seamlessly downloaded and integrated into the system.

The seamless adding, removing and updating of devices and services to the Siemens Smart Home is achieved, to the large extent, by using:

- UPnP (refer to chapter 3.3.4) protocol for device plug & play capabilities and,
- OSGi [34] as a hosting framework (see figure 3.4).
- The OSGi framework is a service-oriented, component based environment that offers standardized ways to manage the software lifecycle.

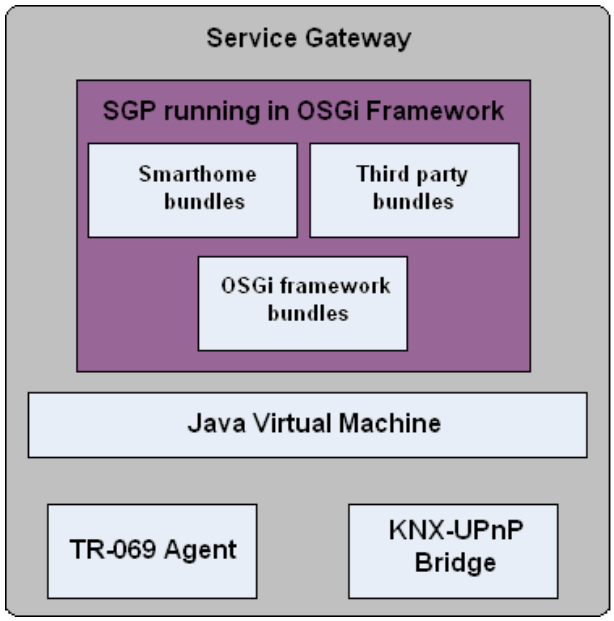


Figure 3.4: Service Gateway Platform

The Service Gateway hardware component is deployed at home. Inside the gateways run several main applications: OSGi framework, Service Gateway Platform, Java Virtual Machine, TR-069 Agent (for remote management and provisioning), KNX-UPnP Bridge (for communication with EIB devices). Inside SGP various OSGi bundles (applications) are deployed: core OSGi framework bundles, Siemens Smart Home architecture bundles and developed by 3rd party services (3rd party bundles).

3.2.4 Service Enabling & Management Platform

The Service Enabling & Management Platform (SEP) resides at the operator's backend and is implemented using EJB technology. The SEP platform has many tasks of which the more important are:

- Subscriber Management (Authenticating users, storing user data)
- Service Management (Gateway configuration, SGP configuration, 3rd party configuration, 3rd party API)
- Remote Access Management (Authenticating users and 3rd parties for remote access)
- Logging, Accounting, service Functions (Trouble Shooting).

From the point of view of service development it is important to stress that many types of services (e.g. Video on Demand) might reside also on the backend. In some cases the application consists of various components written in different technologies and located in distinct places, e.g. EJB for the SEP platform, OSGi bundles for the SGP platform, .NET CF for PDA UI.

3.2.5 User Interface Extensibility Platform

In order to accommodate User Interface on various interaction devices a User Interface Extensibility Platform was designed (UIEP) [66]. The UIEP exists in three forms: the UIEP for web application at the SEP or at the Service Providers, the UIEP for local web application inside the SGP and the UIEP for rich client on the rich client platform.

Web UIEP
The Web UIEP provides a container framework for web applications and provides access to smart home core functions as device and other service access, reception of device/service events, notification broadcasting facility, logging and user authentication. Web UI applications are deployed inside the UIEP during run-time. Updates are also feasible during run-time. The Web UIEP is available as part of the SEP (for web UIs hosted by the Service Integrator), at the Service Provider (for web UIs hosted by a Service Provider) and inside the SGP (for local web UIs for local services or as fallback in case of no network connectivity for hybrid services).

Rich-client UIEP
For each supported rich-client a UIEP is designed. In the current version of the Siemens Smart Home a Pocket PC UIEP based on .NET Compact Framework is available. The SGP supports the different UI technology by the concept of UI adapters. These adapters provide the Smart Service Gateway API for a specific protocol. Current version supports a Web Services protocol adapter for .NET and Java UIEPs. The rich client UIEP provides an extensive development supported by an SDK with container APIs for direct access to available devices and services and core platform functions. Furthermore, a set of UI widgets are provided to ease the implementation of the Smart Home User Interface Guidelines.

3.3 Technology overview

In this section a number of technologies used in the Siemens Smart Home project are introduced, e.g. OSGi framework, EJB, .NET Compact Framework, UPnP.

3.3.1 OSGi

The OSGi [34] framework stands for Open Service Gateway Initiative. The framework is a standardized execution environment for applications and is deployed at the Service Gateway Platform at home. In order to allow the framework for smart and effective management of applications, a set of APIs must be implemented by all applications. These adopted programs are than called OSGi bundles. The framework creates a small layer that enables flexible communication between multiple Java based components and a dynamic discovery of bundles. At the same time the applications are executed in a sandbox, so the running programs cannot harm the environment. For many standard functions like HTTP server, logging and XML, components developed by many vendors are available. Since the OSGi platform needs relatively small computation power it can be deployed on many embedded devices (e.g. car entertainment system, mobile phones). Furthermore, the flexible remote management architecture allows the management of many platforms from one single management domain. The architecture of the OSGi specifications consists of several layers and is depicted on figure 3.4.

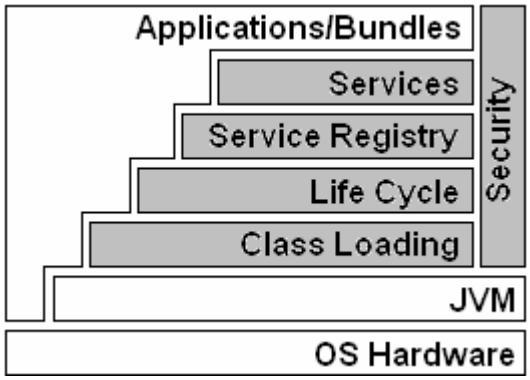


Figure 3.5: OSGi Layers

The picture presents several layers in OSGi specification. The grey layer belongs strictly to core OSGi framework. The functionality of the OSGi can be extended be adding new bundles/applications.

The bottom layer represents Operating System Hardware, where the OSGi framework is deployed. The OSGi specifications are based on the Java Virtual Machine (JVM). All bundles inside the framework are executed inside a single JVM, which enable memory and computation power to be saved.

The next layer defines class loading policies (based on Java plus modularization). The Life Cycle management layer enables the bundles to be dynamically installed, started, stopped, updated and uninstalled. This is done via a special API, which is implemented by all the bundles. The sophisticated dependency mechanism ensures that the life-cycle operations are performed harmlessly to the environment.

The Service Registry layer is the extensive mechanism for dynamic cooperation between bundles. In the OSGi framework the bundles can expose services and use service exposed by other bundles. Once an application is running its services are available via the Service Register. This layer ensures, via special events, the proper operation of the environment when bundles are dynamically added, removed and the services their exposed are still in use by other applications.

The security model is based on Java and Java 2 security model. This overcomes many problems like pointer manipulations, uncheck access to arrays, restricted access to resources, etc. The OSGi

Service Platform separates also bundles from each other. The bundles need to have special permission to cooperate with other applications.

3.3.2 EJB

The Service Enabling and Management platform is based on the Enterprise JavaBeans technology [52]. The EJB is a standard for building server-side component architecture. It simplifies the building of enterprise-class, scalable, secure and distributed components applications in Java. A typical enterprise software solution based on the EJB technology consists of:

- An EJB server and EJB container
- EJB hosted by the container
- EJB clients
- Additional systems, e.g. Java Naming and Directory Interface (JNDI), Java Transaction Service (JTS)

The EJB container is a runtime environment for the beans. It manages the beans
The EJB specification defines a bean – container contract. This is a set of rules that describes how enterprise beans and containers have to be written. In this way, any bean can be deployed in any container, provided that both conform to the specification. In the Siemens Smart Home an open source container JBoss [18] is used. Clients make use of the beans always via an EJB container.

3.3.3 .NET CF

PDA is one of the more important control device types in the Siemens Smart Home. They support rich UI clients, which gives a lot of freedom in implementing an interface for different services. Since .NET Compact Framework (.NET CF)[67] is a common environment for most of the PDAs, it was used to develop of the UI Extensibility Platform (UIEP) running on a PDA.

The .NET CF implements compatible subset of the functionality of the full .NET framework ("lite" version) and it adds some libraries specially designed for mobile devices. The runtime engine sits on the device and is responsible for managing the execution of .NET applications. All .NET applications run under common language runtime (CLR). CLR performs just-in-time compilation of all codes at run time: allocation memory for new object; garbage collection, etc. (similar to java runtime environment). The CF uses a new implementation of CLR that runs effectively on embedded devices.

The development process in .NET CF is to large degree similar to the one in the full .NET framework. The programmers can save on learning process, since they can profit much from their knowledge and experience. The Integrated Development Environment for .NET CF provides additional tools especially for embedded devices programmers, e.g. connection to and debugging on a remote device, embedded device simulators.

3.3.4 UPnP

UPnP [56] stands for Universal Plug and Play. In the Siemens Smart Home it is used to integrate various devices. The devices should support the UPnP protocol, either directly or via a bridge, e.g. for the EIB technology there is a EIB-UPnP bridge running on the gateway. The SGP uses this protocol to control all devices available in a smart home.

UPnP technology extends the Plug & Play concept and defines architecture for pervasive peer-to-peer network connectivity of devices and services in the home environment (PC, PDA, white

goods, electrical devices …). It is designed to bring easy-to-use, standards-based connectivity to ad-hoc or unmanaged networks. UPnP is based on TCP/IP and HTTP technology.

In UPnP there are two types of devices: control points and controlled devices. An example of a control point could be PDA, PC or phone. The control devices contain services which define the functionality offered by an UPnP device. UPnP devices and services are standardized by UPnP Forum. The device is then described in DCP (Device Control Protocol), e.g.:

- BinaryLight V1.0 – description of an UPnP device binary light,
- DimmableLight V1.0 – description of an UPnP device dimmable light,
- SwitchPower V1.0 – description of an UPnP service Switch Power.

UPnP protocol has several steps (see figure 3.6):

1. Addressing – the devices gets a valid IP address,
2. Discovery – the control point finds controlled devices,
3. Description – the control point receives device and service description in XML,
4. Control – using SOAP the control point can invoke actions of services,
5. Eventing – using GENA (General Event Notification Architecture) UPnP devices inform the control points about the change of the state variables,
6. Presentation – using HTML or WML the presentation page can be presented for user control.

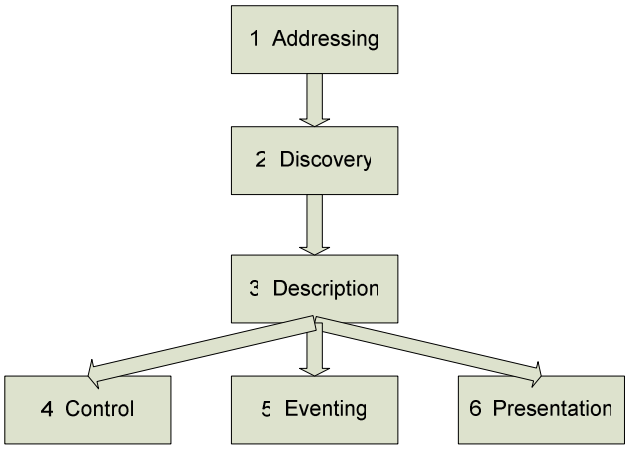


Figure 3.6: Steps in UPnP Layers
The picture presents 6 steps defined by the UPnP protocol.

In the Siemens Smart Home the UPnP control point, which is located at the SGP, must know all the UPnP devices available in the network. For this reason, it sends a search message to the network (see figure 3.7). All available UPnP devices will answer with a response message (discovery phase). In the next step, called description, the devices and belonging to them services belonging to them send more detailed information about their properties (e.g. types of services, action names, and parameters). After that, the device is ready to be commanded by a control point. The control point can fire action on various services and listens to events.

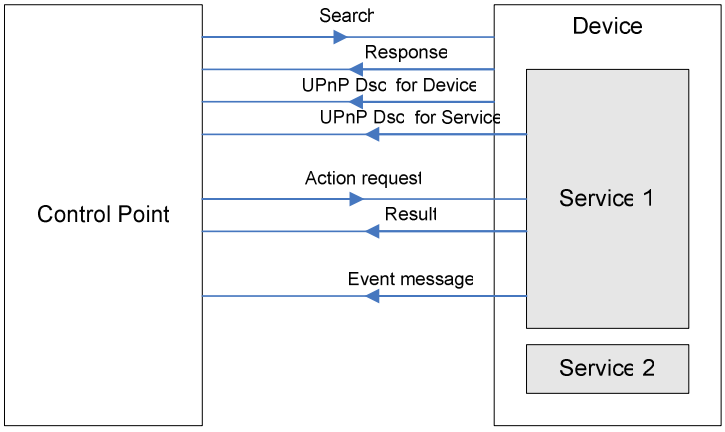


Figure 3.7: UPnP Control Point- Device communication

The picture presents the UPnP communication between a control point and a device. The device contains one or more services.

3.4 Siemens Smart Home services

This part is an introduction to services in the context of a smart home. Based on previous studies an explanation of a service, service types and classification method is given.

3.4.1 Service Types

Occupants of a smart home benefit from various services, e.g. a security service, a device control service or a scene controller service. These services can be locally specific, network specific or can be a combination of both. A device control, which gives a user a possibility to control home appliances from various control devices, is an example of a local service. In this case, the service is restricted to a specific house and does not have any connection to outside systems. On the contrary, weather information is a typical example of a network service. A user can get only useful information from the network and is not able to affect directly the state of the smart home. A standard example of a hybrid service is security. Inhabitants have, on the one hand, a certain amount of control over security system at home, e.g. arming and disarming. On the other hand, the system is connected to outside legacy system in order to raise an alarm in case of abnormal situation.

The variety of services enforces a use of different service architectures during a development process. In a distributed and heterogeneous smart home environment a service implementation is composed of various elements deployed in different parts of the system.

As an example let us consider two services: weather information and a device control. Both of them are of different types, thus the developed components are different. Weather information, which is a network service, will have a part running in the backend server. The part will be responsible for gathering information from legacy systems and providing the information for the smart home inhabitants. The gateway in the home has no functional role in this scenario. The device control service enable users to control home appliances from a PDA, PC, STB or cordless phone. Because there is no need for this service to communicate with any outside resources, the service will be mainly implemented as an OSGi bundle running on the gateway.

When a developer wants to introduce a new service, he needs to identify the service type and decide which components to implement. For somebody, who is new in smart home architecture, this process can be long and difficult. Having this in mind a clear classification schema is needed.

3.4.2 Previous studies on service types

A good classification of smart home services was done by Johannes Wechsler [64]. At first he gathers together many possible smart home services, e.g. Medical Data Recording, Energy Savings, and Family Calendar. In a second step some common properties between those services are investigated. Based on those properties he defines eight service classes:

- Manual Device Control
- Autonomous Device Control
- Extended Manual Device Control
- Extended Autonomous Device Control
- Information
- Web-Based Services
- Network Integration of Local Applications
- Media Streaming from Home
- Media Streaming from Remote

Each of the above service classes has a well defined architecture, which tells a developer exactly what components are needed and where they must be deployed. With the service classes there is no need for a developer to investigate the whole smart home architecture in depth. This approach can save much time and is less error prone, especially when a developer is new to smart home environment. A new service must be classified to one of the service classes. In this case, it is important to provide a proper classification method.

3.4.3 Classification method

A classification method for smart home services proposed by Johannes Wechsler in his work [64] assumes that each service can be described by a set of 18 characteristics, e.g. interaction with actuators and/or sensors, size of repositories for generated data, amount of information transferred to external entities. Each of these characteristics has weight and defined set of values. A programmer has to choose values (from the given set) for all the characteristics, e.g. *desired availability at home* can be set to >=99,99% (from the set of three values: >=99,99%, >=99,9%, >=99,0%). After this a special scoring algorithm developed by J.Wechsler will calculate scores for all the service classes. The service class which gets the higher score is the most suitable for the given service. However, the classification method is not perfect and is can happened that a developer will be not satisfied whit the chosen service class. Therefore it is important to make a choice from the set of service classes which have the scores.

3.4.4 Classification method evaluation and relation to Siemens Smart Home

The above described classification method presents a general approach. It is based on a standard smart home architecture, not specific for any proprietary solution. Applying the classification to the Siemens Smart Home needs some verification and remarks.

Basically, in the Siemens Smart Home there are three service classes defined: local services, network services and hybrid services. The first class corresponds to the services whose functionality is restricted to home, the second comprise services whose functionality is located in the network outside home, and the third is the combination of both.
Wechsler's classification is more detailed and assumes the existence of some additional components as data storages, media renders, content generators, etc. Those components also exists in the Siemens Smart Home, however they are not taken into consideration by service classification applied there.

This means that the tool provided by Johannes is suitable for classification of services, since they can be easily mapped to the Siemens Smart Home class services. Moreover, these studies are based on the first release of home platform. Siemens Smart Home will be further developed in the future and the next version might assume more service classes.

The table below presents how the service types from Wechsler's master thesis can be mapped to the Siemens Smart Home types. On the left are service types proposed by Wechsler and at the top are the ones used in the Siemens Smart Home. The cross means both service types correspond to each other.

		Siemens Smart Home service types		
		Local	Network	Hybrid
Service types based on Wechsler's studies	Manual Device Control	x	-	-
	Autonomous Device Control	x	-	-
	Extended Manual Device Control	-	-	x
	Extended Autonomous Device Control	-	-	x
	Information	-	x	-
	Web-Based Services	-	x	-
	Network Integration of Local Applications	x	-	-
	Media Streaming from Home	x	-	x
	Media Streaming from Remote	-	-	x

Table 3.1 Service mappings

4 Service development

At the beginning of this chapter a general software development process, which consists of various activities, is presented. In the next section this process is evaluated against the development of services in a smart home environment (in particular Siemens Smart Home). Similarities and differences are shown.

4.1 Software development process

Software engineering deals with developing high quality software which is delivered on time and within budget [4]. In order to achieve this goal, software engineering is described by various high-level activities such as modelling, problem solving, knowledge acquisition and rationale management. During modelling one needs a good understanding of the environment and a system one wants to build. This knowledge is then represented in an abstract model, which is focused only on relevant issues. The next activity, problem solving, includes formulating and analysing a problem, searching for possible solutions and choosing an appropriate one. Developers experiment here with various technologies, try to reuse well known pattern solutions, test the system against functional and non-functional requirements, etc. As a result, the system develops incrementally to the final version.

The presented activities describe software engineering from the high-level perspective. Talking about methods to develop software, lower-level development activities are more interesting. This includes:

- requirements elicitation,
- analysis,
- system design,
- object design,
- implementation,
- testing.

The studies in this work do not assume any software development model (e.g. waterfall, spiral model). The tools and activities presented it the master thesis can be used with any development model, e.g. waterfall model.

Requirements elicitation
During requirements elicitation the purpose of the system, functional and non-functional requirements are to be defined. This process is normally carried out together with a client, who provides system information. At the end, the use case model is validated via usability tests. Some problems in the model can be found by letting the user investigate the system. The result of this activity is a description of a system in terms of actors and use cases.

Analysis
The next phase, analysis, produces a model of the system that is correct, complete, consistent, unambiguous, realistic and verifiable. Developers transform the use case model from the requirements elicitation to the object model, which completely describes the system. The analysis model is composed of three individual models:

- functional model – represented by use cases and scenarios
- analysis object model – represented by class and object diagrams
- dynamic model – represented by state charts and sequence diagrams

System design
The main goal of the system design phase is to decompose the system into smaller subsystems that can be developed by different teams. At this stage detailed strategies for building the system should be designed (e.g. hardware/software platform, access control policy …). Developers should also create a deployment diagram representing the hardware/software mapping of the system.

Object design
The object design is more detailed as system design. Here the main working unit is a single subsystem, for which all objects and interfaces should be described. Developers investigate possible pattern solutions and off-the-shelf components for subsystem usefulness. The resulting model can be than restructured in order to ensure better extensibility or understandability, and optimized for higher performance.

Implementation & testing
During implementation a working source code is produced. In the case of a heterogeneous environment this process can be complicated due to integration issues. However, much of the interoperability problems can by overcame by actions performed at the previous stages. During implementation a good testing method is critical. In complex system there are different tests, based on the project stage:

- Usability testing – tries to find faults in the user interface design of the system
- Unit testing – tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
- Integration testing – the activity of finding faults when testing the individually tested components together.
- Structural testing – culmination of integration testing involving all components of the system.
- System testing – tests all components together (as a single system). This includes fictional (requirements), performance, acceptance and installation testing.

Deployment
Deployment includes installing, setting up, testing and running software on the customer side.

Maintenance
This process involves changes to the software in order to correct defects and deficiencies found during usage, as well as the addition of new functionality to improve the software's usability and applicability. During this process various software versions and configurations have to be managed.

4.2 Service development

Service development is a special case of the software development process described above. However, the difference is that in case of a smart home there is no need to build the whole complicated system. Much of the hardware/software infrastructure considerations and technology decisions have already been made. The whole system is actually already built and should operate without errors. A new service must be designed according to the given smart home architecture, which enables seamless integration into the working system. Therefore the size and amount of software development activities described in the previous chapter are decreased.

As an example let us consider a mood management service. The service should give a user the ability to fire previously programmed moods, e.g. romantic, living home, enter home. Inhabitants make use of the service via designed UI. In case of the Siemens Smart Home various UI platforms are already defined (e.g. rich and thin clients). A developer should only program a mood management UI plug-in for a PDA in case of the rich platform, or design a web page in case of this client. In both cases the architecture, programming language, technology and method for UI are already de-

fined. Furthermore, the mood management makes use of the services which are already available, e.g. lights and blinds control, media playing. Application logic consists mainly of the integration of those services. In this case a natural place for the application logic is a home gateway, since it has access to all devices in the home. Again, in Siemens Smart Home the architecture of the gateway is defined and the freedom of the developer is restricted. Of course, this is a relatively simple example. One can think about more sophisticated services, like health care service, where the amount of work to be done is significantly larger.

On the figure 4.1 a use case diagram of service development process is depicted. The development starts with requirements elicitation which aims at defining a new service. During analysis a more detailed model of an application is constructed. During the next phase, service design, service architecture is decomposed into subsystems. In the object design phase a detailed model of each subsystem is created. The implementation includes such activities as programming service logic, programming UI, integrating legacy components, etc. Different tests are performed parallel to service implantation. Configuration management is also needed due to the large variety of users and smart home providers. Different users have their own preferences (e.g. installed services) and various providers might enforce different platform polices (e.g. payment). After the software is ready it needs to be deployed at the user's home. This includes installing and configuring all home devices and hardware / software components. Moreover, due to the large variety of integrated technologies in smart home, the software is very likely to change over time. Therefore, a version management system, which ensures the proper cooperation of all software components, is needed. In service development process a number of actors are involved, such as system/service analyst, service developer, end user, tester, etc. Many different actors correspond often to many people and entities involved in the development process. As a result, the communication process between those actors is very important.

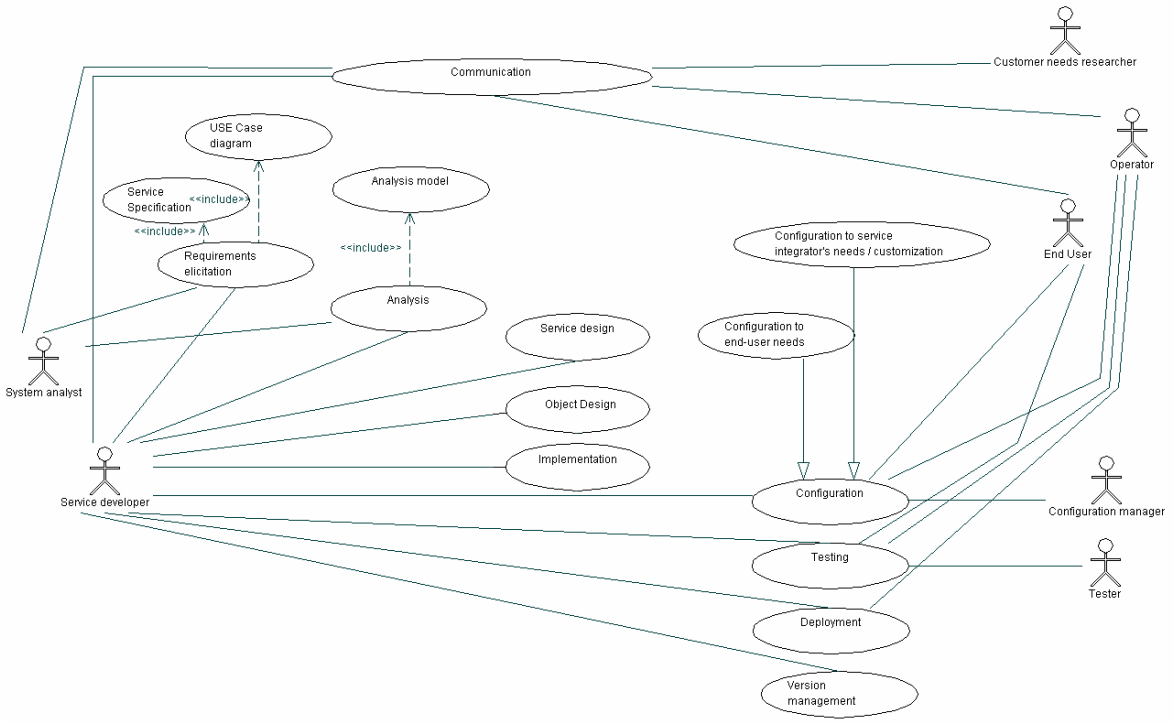


Figure 4.1: Service development – general use case diagram
The picture presents general use case diagram of service development.

4.2.1 Requirements Elicitation

The main purpose of requirements elicitation is to define the purpose of the system. In a smart home environment this process should start with market research to find out answers for some questions like:

- What kind of services do inhabitants find interesting?
- Are they going to use it?
- Are they going to pay for it?
- How much they are prepared to pay?
- …

In this phase a system/service analyst must communicate with many other actors: end user, operator, customer needs researcher, etc… End users and customer needs researchers should provide the input about the type of a new service.

When a use case diagram for a new service is constructed, it should be examined by usability tests. However, since the user have no possibilities to play with a real application, the true evaluation can be impossible.

4.2.2 Analysis

The next phase, analysis, produces a model of the system that is correct, complete, consistent, unambiguous, realistic and verifiable. Developers transform the use case model from the requirements elicitation to the object model, which completely describes the system. The analysis model is composed of three individual models:

- functional model – represented by use cases and scenarios
- analysis object model – represented by class and object diagrams
- dynamic model – represented by state charts and sequence diagrams

4.2.2 Service Design

The main goal of the service/system design phase is to decompose service architecture into subsystems in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, etc.

This phase is significantly smaller in comparison to the development of other large software projects. In the case of an already designed smart home solution, like Siemens Smart Home, many hardware/software decisions and policies are already made.

Service design use case diagram for the Siemens Smart Home is depicted on figure 4.2. The design consists of activities:

- Service logic design – a developer can make use of previously designed services or use services form 3rd parties (e.g. Web Services available in the Internet)
- Integration of legacy components – some device types available in a smart home might have a proprietary communication protocol. There must be a software or hardware component, which makes devices available as an UPnP devices
- UI design (device specific) - design of UI for different types of devices and support for different models (rich and thin client). The UI designer can use a widget library which provides the programmers with common useable User Controls (e.g. icons, buttons).

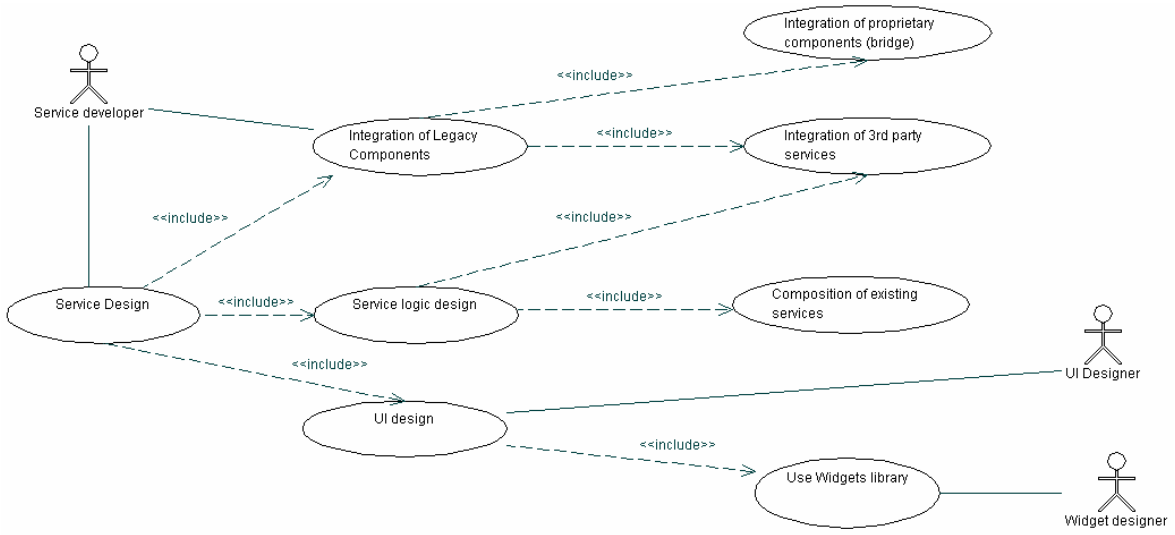


Figure 4.2: Service design
The picture presents use case diagram of service design activity.

4.2.3 Object Design

Object design aims at creating more detailed models of the subsystem from the previous step. The tasks in this case are general of object design and have already been described at the beginning of this chapter.

4.2.4 Implementation

During implementation a working source code is produced. In the Siemens Smart Home this process includes three main activates:

- Integration of legacy components – for this purpose a UPnP Bridge is implemented
- Service logic implementation – programming a new code as well as integrating already design services
- UI Implementation – implementing rich and thin client for various interaction devices (e.g. PDA, mobile phone, PC, …)

During implementation a developer is supported by a number of tools. Integrated Development Environments are currently a standard when developing any software in high level languages like java, c. c++, c#. For the Siemens Smart Home various code generators can be used. This can save much time and decrease an error ratio.

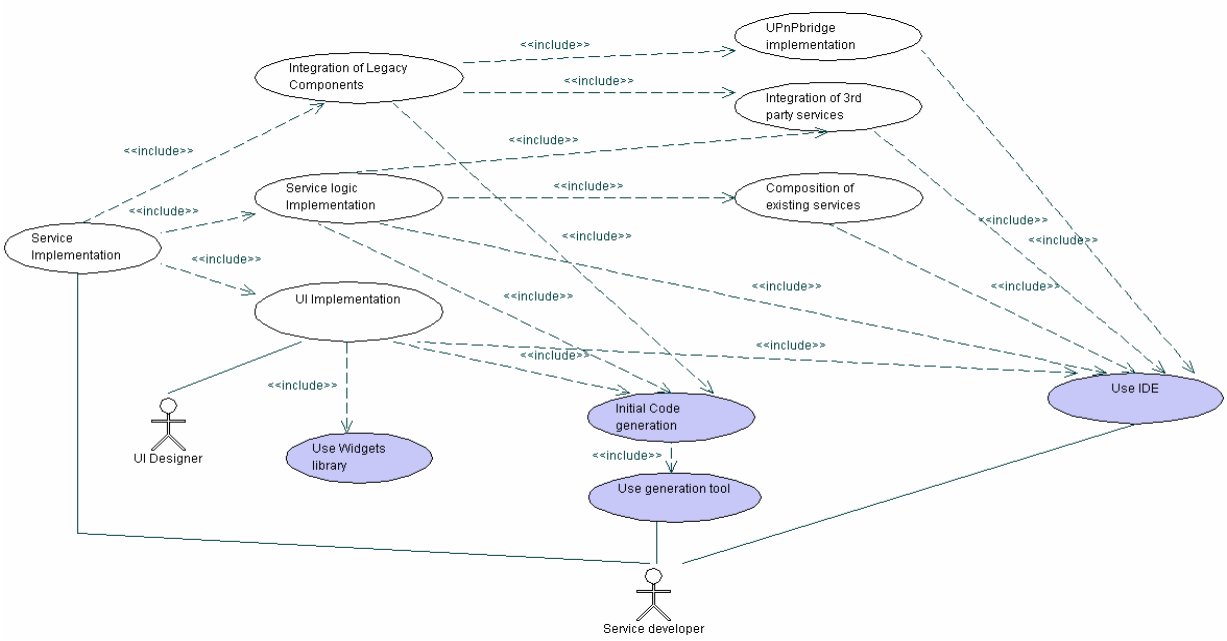


Figure 4.3: Service implementation

The picture presents use case diagram of service implementation activity. Grey use cases means that the activity includes the use of support tool or component (e.g. generator).

4.2.5 Testing

Testing aims at finding differences between the system and its model by executing the system (or parts of it) with sample input data sets. It is important to discover as many faults as possible, so the software can be fixed before the delivery.

On the figure 4.4 a use case diagram describing various test activities in the Siemens Smart Home is presented. In heterogeneous and distributed environments it is important to test all the "service building blocks" before integration. Thus during unit test a number of components are tested: Service Faces (Web UIEP architecture), UPnP bridge (integration for legacy devices), OSGi bundle (service logic running on the gateway) and UI PDA plug-in.

The next step is to perform the integration tests. This includes the examination of interaction between:

- UI on the PDA OSGi bundles on the gateway,
- Bridge application and real devices with OSGi bundles on the gateway.
- Gateway at home and backend system

At the end all the service components must work together without problems.

Moreover some functional, performance and usability tests should be preformed. It is very important for the system to support remote debugging and error reporting. Bugs and errors in the smart home environment can occur later when the system and set of services are already deployed at many customer houses. It is crucial to find and solve the problem as quickly as possible.

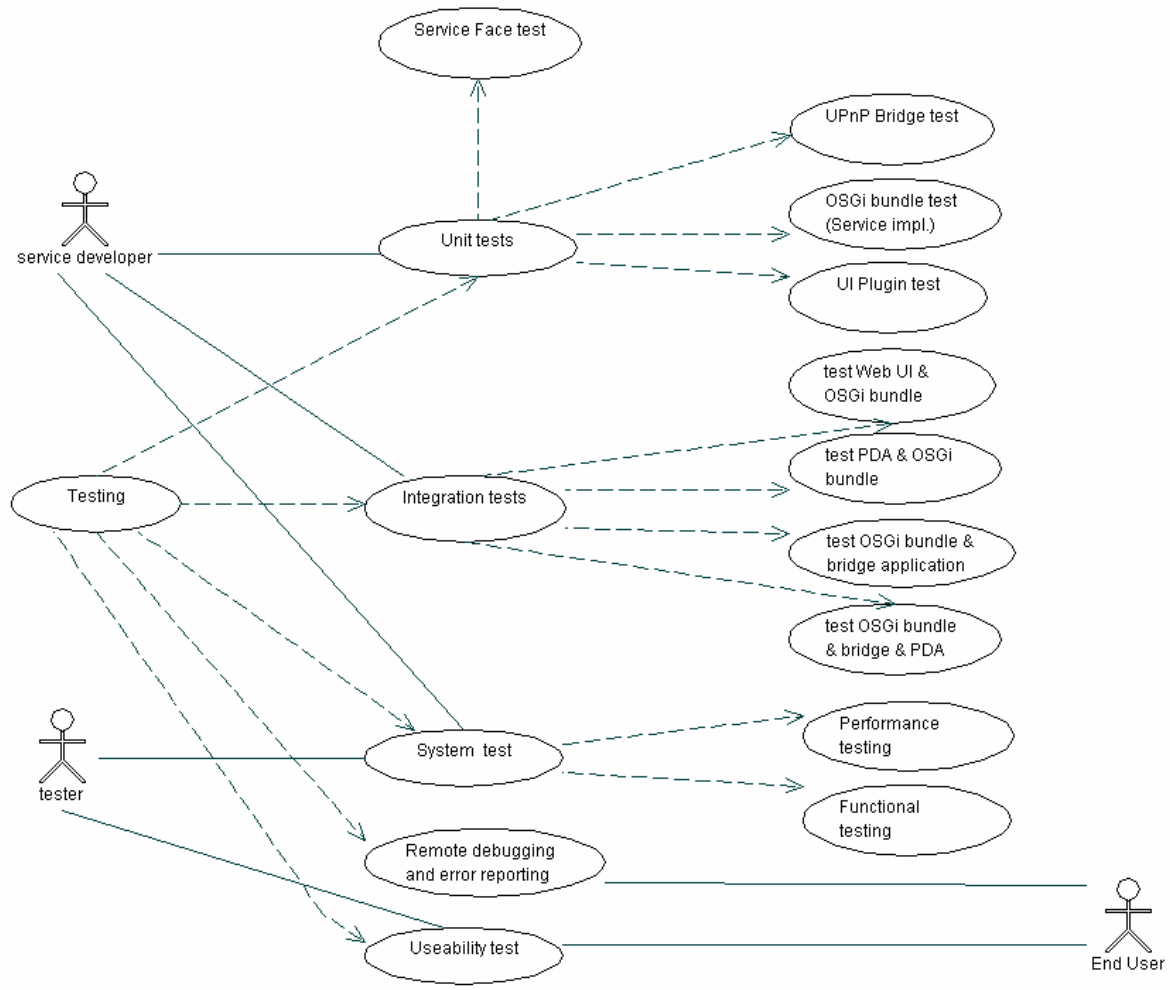


Figure 4.4: Testing in Siemens Smart Home – use case diagram

The picture presents use case diagram of service testing activity in the Siemens Smart Home. Unit tests check the proper functionality of individual components while integration tests check their proper cooperation. The system test checks the whole system in terms of performance and functionality.
Legend: dashed line - <include> stereotype

4.2.6 Configuration management

A large variety of users and smart home providers demands good configuration software. Different users can have different set of services, devices at home, etc. Therefore the individual preference data should be stored in the backed server. The backed server manages many gateways and can be used to recover the individual smart home system in the case of failure.

Moreover, each smart home provider has his own policies and integration needs, which can concern billing information, supported services, devices, etc. Since among various providers this data can differ, different variants of the Siemens Smart Home system should be easily built.

4.2.7 Version management

In a smart home one deals with a lot of different technologies, e.g. gateways, home appliances, control devices, etc… Given such variety of components, it is very likely for them to be replaced

quite often by new versions. This usually requires updating software. Therefore, proper version management polices and a supporting tool is needed.

4.2.8 Deployment

In the Siemens Smart Home the deployment process is mostly automatic. New services are deployed from a backend server via a management link. However, in the case of installation of new devices, a technician might be need for the integration of new appliances into the smart home. In current Siemens Smart Home solutions an expert is needed to set the EIB network and SiRoute devices.

European Installation Bus – EIB [12]
The European Installation Bus is designed as a management system in the field of electrical installation for load switching, environmental control and security, for different types of buildings. Its purpose is to ensure the monitoring and control of functions and processes such as lighting, window blinds, heating, ventilation, air-conditioning, load management, signaling, monitoring and alarms.
A technician makes an EIB installation from EIB devices connected via twisted pair and communicating with each other via the EIB protocol. Depending on the function, there are several types of EIB devices (e.g. binary input/output, dimmer, temperature controller, power switch, switch button, etc.).

SiRoute [45]
Fire and security products from Siemens Building Technologies are fully wireless and bidirectional system based on EasyRouting radio technology. *The patented telegram routing algorithm EasyRouting passes on the information from the wireless participants to the radio controller in the gateway and vice versa. The communication works by passing telegrams from one participant to the radio controller via other participants if a direct radio communication is not possible. In the case of radio disturbances EasyRouting automatically searches for an alternative route between the participant and the radio controller [45].* A specialist has to install all the devices at home and configure the system using SiRoute Installer Tool.

5 Current tool support

In this chapter a number of software tools are briefly described. At first applications which are helpful in general software development (e.g. IDE, UML tools) are depicted. Second smart home specific tools are listed, e.g. OSGi plug-ins, Engineering Tool Software. In the last step the FXL Framework is presented. It importance, advantages and disadvantages in the context of smart home are discussed.

5.1 Standard tools

The best practices of software engineering have been investigated for many years at universities and research institutes. This results in better and better methodology for various kinds of software development like: large heterogeneous, distributed over network systems, or small applications for embedded devices. To support this process a number of tools have been developed. Those tools can be used for some stages of service development.

5.1.1 UML (for requirements elicitation, analysis, service and object design)

The Unified Modelling Language (UML) [13] is a current standard for specifying, visualizing, documenting and modelling software architecture. This graphical language defines several diagram types, which describe different aspects of software system, e.g. used case, class and interaction diagrams. There are already many stand alone tools and plug-ins for popular Integrated Development Environments, like:

- Borland Together Technologies [55]
- Microsoft Visio [61]
- Rational Rose XDE Developer [40]
- EclipseUML [33]
- Visual Paradigm for UML [59]
- MagicDraw UML [30]

Different kinds of UML diagrams are widely used in such phases as: requirement elicitation, analysis, service design and object design.

5.1.2 Configuration and version management

A smart home environment integrates a lot of different technology and is addressed to a broad spectrum of customers. Therefore a good configuration and version management software is a must. There are already many configuration management tools, commercial as well as free, available to developer. Some of them are:

- RCS, Revision Control System [41] – a free tool, controls repository storing all versions of the configuration items; does not support the concept of branch
- CVS, Concurrent Version System [8] – also a free tool, extends RCS with the concept of branch
- Subversion [49] – a free tool, the main idea was to build a version control system that is a compelling replacement for CVS
- Perforce [38] – a commercial tool, similar to RCS and CVS
- ClearCase [6] – a commercial tool, which additionally supports the concepts of CM aggregates and configurations.

5.1.3 Integrated Development Environments (for implementation and testing)

Nowadays developers write most of the code in IDEs. They provide many convenient mechanisms and tools for programming faster, fewer errors and better quality applications. In the case of the Siemens Smart Home there are two main software technologies used:

- J2SE for gateway applications and J2EE for backend system,
- .NET CF for UI Extensibility platform running on the PDA.

Eclipse
There are many IDEs supporting writing code in java, e.g. NetBeans, JBuilder, IntelliJ IDEA. However, most of the code for the Siemens Smart Home was developed using Eclipse [10]. This open source software provides good sets of tools and allows an easily integration of 3rd party plug-ins. This makes it easy to extend the platform by already written application on the one hand, and allows to write own, specialized extension on the othe.

Visual Studio
Using .NET CF for writing User Interface Extensibility Platform on the PDA implies the choice of Microsoft Visual Studio 2005 [60]. It supports writing code in C# (a language of a UI plug-in) and provides some debugging tools, e.g. a PDA simulator that enables to test applications before deploying them on the real PDA.

5.1.4 Ethereal (for testing)

Ethereal [11] is free to use in its basic but powerful form, a network protocol analyser. Many technologies and different network protocols sometimes cause interoperability problems. Therefore, it is very useful to analyse the traffic in smart home environments on the level of data link layer and be able to inspect single bytes.

5.1.5 Profiler (for testing)

A profiler is a performance analysis tool that measures the behaviour of the program as it runs, particularly the frequency and duration of functions call (www.wikipedia.org). Developers can use this tool e.g. to check whether the non functional requirements are met, or if there is a memory leak in an application. Two example profilers from the market:

- YourKit [70] – a profiler for Java and .NET fro YourKit company
- JProfiler [20] – only a Java profiler from ej-technologies

5.1.6 Unit Tests (for testing)

The unit test should be performed on each piece of code before starting any integration activities. In this method for each function, method and module test cases should be written and run separately from each other (probably by using mock objects). Some of the unit test frameworks available are:

- JUnit [21] – an open source regression testing framework for Java programmes, it was written by Erich Gamma and Kent Beck,
- NUnit [32] – a free unit testing framework for all .NET languages; the code written in C# framework is based on the JUnit concepts,

5.1.7 Build tools (implementation tool)

Build tools, which automate software build processes can be very useful when developing large project consisting of many distributed objects. Some tools available on the market:

- Apache Ant [1] – a free java based tool which read data in an XML format
- NAnt [31] – a free tool similar to Apache Ant for .NET framework

5.1.8 Logging (for debugging and maintenance)

Good logging is a must for almost every application. Especially in distributed, heterogeneous environments, like a smart home, it is very useful for remote debugging and maintenance. There are many logging frameworks available for various platforms. Here I present three of them:

- Log4j [27] – a good logging framework available free from Apache Software Foundation. It enables to change the logging behaviour at runtime (an external property file) change the target log output (file, remote server, etc.)
- Standard Log Service of the OSGi specification [34] – provides basic logging capabilities
- Pax logging [37] – extends the standard logging interface in OSGi specification with additional interfaces. Defines its own API but supports Log4J and Jakarta Commons Loggings APIs
- Log4net [28] – an adaptation of the Log4j framework for .NET environment

5.1.9 Soap and Web Services testing

The Service Gateway Platform (at home) communicates with a PDA and a backend service via web services. A lack of communication due to the false format or content of a SOAP message is possible. Therefore, some SOAP and Web Services testing tools are needed. Some examples of such tools are:

- soapui [47] – an open source tool for testing web services over HTTP
- WebInject [62] – a free tool for automated testing of web application and web services
- Parasoft SOAtest [36] – a commercial tool simplifies SOA development, automates client/server functional testing, regression testing and load/performance testing.

5.2 Smart home domain specific tools

In this part some tools relevant to smart home domain are presented.

5.2.1 UPnP tools (for implementation and testing)

There are a number of UPnP SDKs and tools available form various companies. The full list is on the UPnP forum web site: http://www.upnp.org/resources/sdks.asp. Two examples from the list are briefly described below.

Siemens Plug & Play – Software Packages [44]
Siemens UPnP SDK from Siemens provides a full implementation of UPnP technology protocols in Java (see figure below). The implementation of an UPnP generic control point allows the basic testing UPnP devices. A user can discover all UPnP devices in the network, fire action on them and see incoming events. It includes also a Device Builder, which is a stub generator for OSGi and UPnP device and control points.

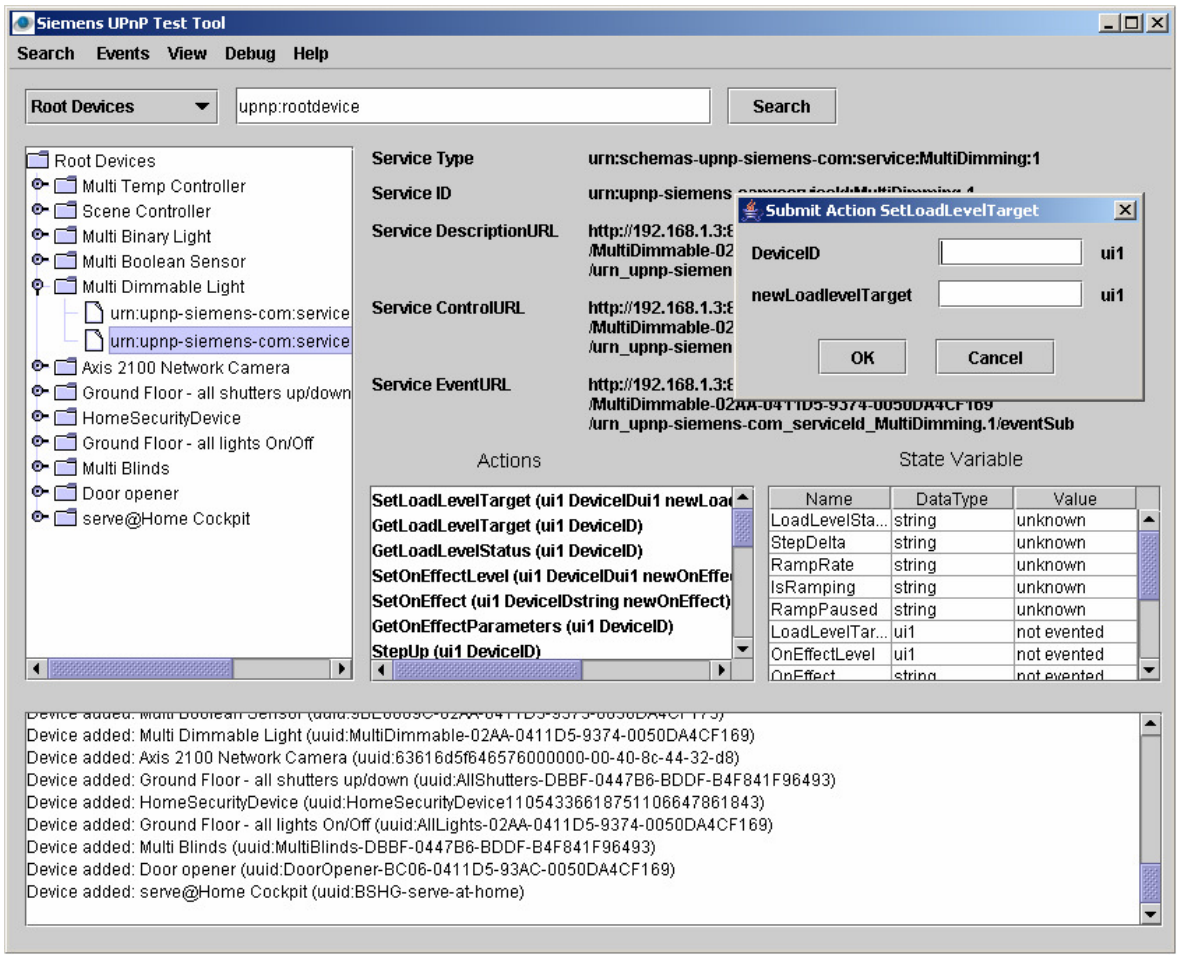


Figure 5.1: Siemens Generic Control Point

The picture presents a screenshot form Siemens UPnP Generic Control Point. On the left various UPnP devices currenty available in the network are visible. A user can see the UPnP services each device exposes and fire action of particular service. As an example a SetLoatLevelTarget action was fired and a user is asked for parameters. It is also possible to monitor incoming events from UPnP devices (e.g. when somebody switches on the light manually an event should be generated).

Intel Tools for UPnP Technologies [17]

The Intel SDK for UPnP was written in C++ language and it provides some additional useful tools. The Device Scriptor allows automated tests on UPnP devices by creating scripts that run against the set of target UPnP services (e.g. test various scenarios on UPnP devices or interaction between them). The Device Sniffer monitors all UPnP broadcast traffic and enables solving SSDP interoperation problems between UPnP devices and control points. The Device Validator can validate and provide a suite regression tests for UPnP devices. At the end the Service Author tool provides an easy way to produce new UPnP services. A user uses GUI to create services, actions, parameter, etc. and the tool generates the UPnP XML description automatically.

5.2.2 OSGi Eclipse plug-ins (for implementation)

Several tools which support development of OSGi bundles are available on the market. In the Siemens Smart Home most of the service software runs as OSGi bundles on the Service Gateway Platform at home.

Knopflerfish Eclipse Plug-in
The Plug-in created within the confines of Knopflerfish project [23] and is free to plug-in for Eclipse (figure 5.2). It supports creating new bundles, editing bundle manifest, running and debugging bundles in a OSGi environment.

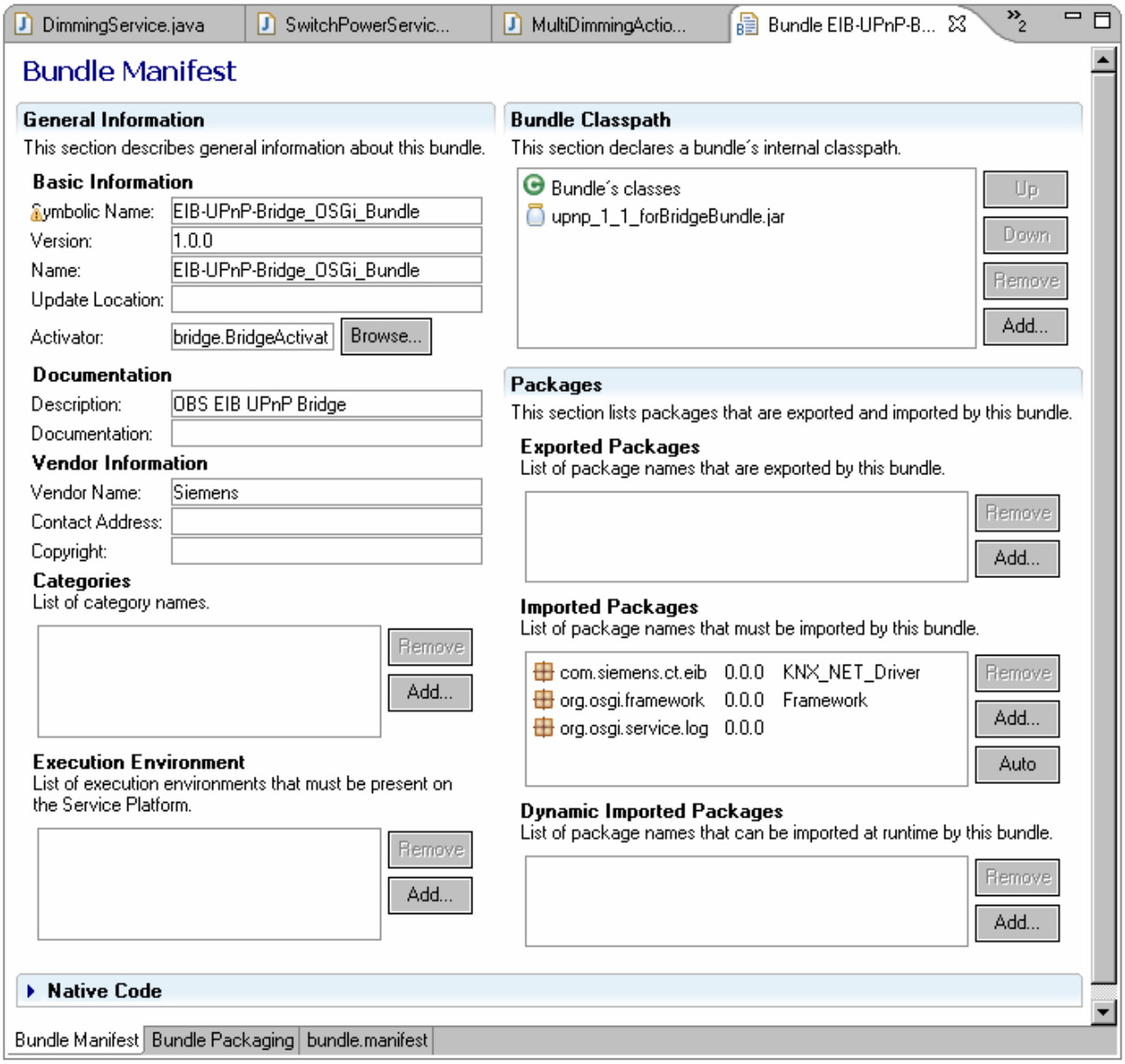


Figure 5.2: Knopflerfish Eclipse Plug-in
The picture presents a screenshot from the OSGi Knopflerfish Eclipse Plug-in. On the picture GUI for manipulating OSGi bundle manifest.

ProSyst OSGi Eclipse Plug-in
The plug-in from ProSyst [39] is a commercial tool supporting the development of OSGi bundles with Eclipse platform. Additional to the previous plug-in futures, it provides also a connection to OSGi frameworks and allows their administration with console commands via the Remote Control utility.

5.2.3 Engineering Tool Software ETS3

The Engineering Tool Software ETS™ 3 [12] is a Windows tool used for the planning, commissioning and diagnostics of an EIB/KNX installation (see figure 5.3). It contains the configuration of the installation (installed EIB devices, device properties, communication addresses, group addresses …). In ETS3 one can also correlate functionality of various devices, which is made by grouping different communication objects with a group address. The configuration of an EIB installation is made only once, after the EIB infrastructure has been laid down.

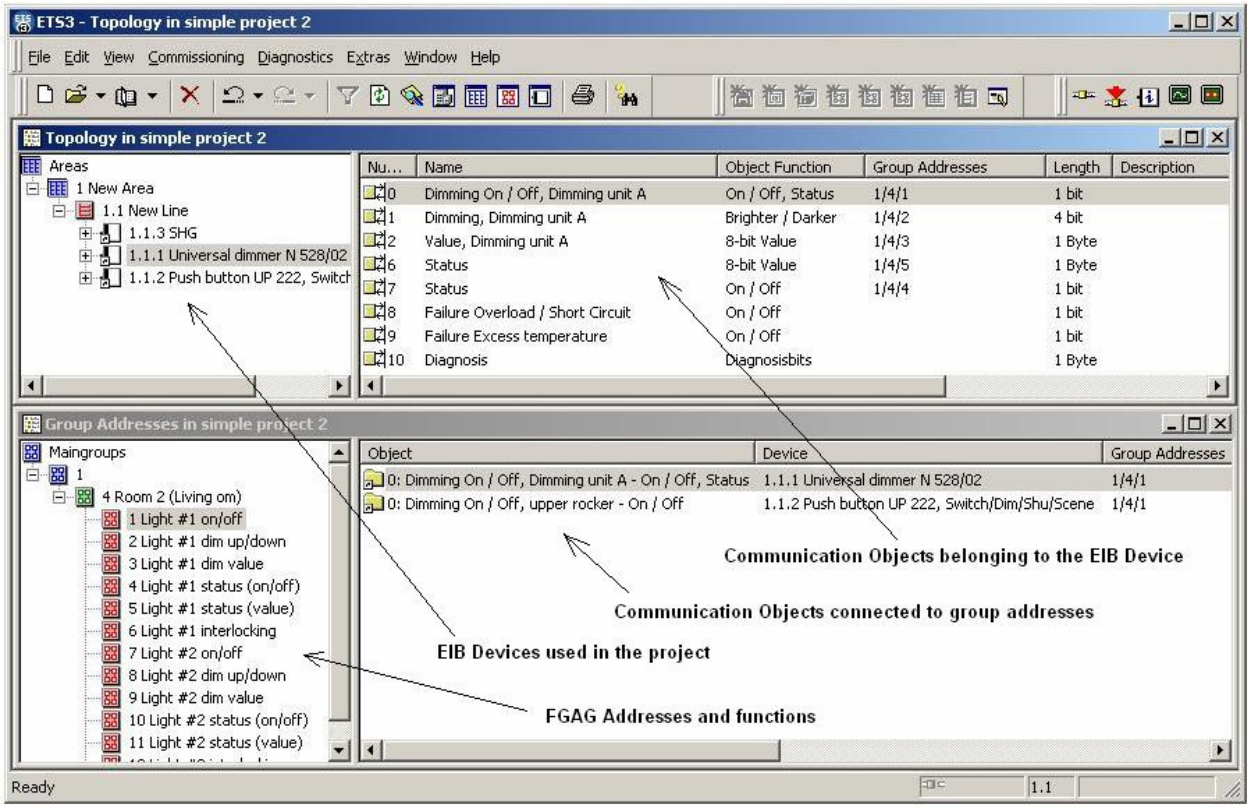


Figure 5.3: Screenshot from ETS3

This picture presents a screenshot form Engineering Tool Software ETS3. In the left upper window are some ETS devices added to the project (e.g. Universal dimmer). On the right upper side one can see communication objects of a currently selected EIB device. On the bottom there is a group address topology together with associated communication objects.

5.2.4 ETS3 plug-in: Smart Home Gateway

All the EIB devices (dimmers, switches, rollers, etc.) have to be available as UPnP devices in the home network. Since the current EIB protocol does not support UPnP protocol, the EIB network has to be mapped by an EIB-UPnP Bridge to the UPnP network. The Bridge software uses for its purpose an XML configuration file, which provides special mapping data.

If the ETS3 project follows the FGAG specification [50], a Smart Home Gateway plug-in can be used to automatically generate the XML configuration file for the EIB-UPnP-Bridge. *The Functional Group Addressing Guidelines (FGAG) is based on a model of predefined group addresses for specific functions. FGAG covers functions for lighting, shading, HVAC, security and energy management. The model describes functional entities for*

various rooms and contained devices (e.g. Room3/Kitchen - Light #1 on/off), as well as global functions (e.g. Whole House – Solar Protection South Level 1) [50].

5.2.5 SiRoute Installer Tool

SiRoute Installer Tool is a Windows and web browser based application for configuration installation and maintenance of SiRoute security system.

5.2.6 Service Architecture Chooser

Service Architecture Chooser is an Excel tool developed by J. Wechsler for his master thesis [64]. The tool makes use of studies performed by J. Wechsler, i.e. classification of smart home services and classification method. In the *Service Architecture Chooser* a developer is asked to input a number of values for some variables. Based on these values a ranking of best suitable service classes is made. A class which suits a service the most is given the most scores. The service class type corresponds to the specific architecture. A developer can view all the classes, their scores and corresponding architectures to review and eventually change proposed architecture.

5.3 Tool – summarise

The table 5.1 below summarises the tools presented until now. Columns represent different activities in service development process, such as requirements elicitation, implementation, deployment, etc. Rows of the table divide the tools to standard and smart home domain specific. The tools are assigned, according to their usability, to various development processes.

	Requirements elicitation, Analysis, Service Design, Object Design	Implementation	Testing	Configuration	Version management	Deployment
Standard tools	UML					
				Configuration and Version Mng.		
		IDE				
			Ethereal			
			Profiler			
			Unit Tests			
		Build Tools				
		Logging				
			SOAP & WS			
Smart home domain specific tools		UPnP Device Builder	UPnP CP			
		OSGi Eclipse plug-ins				
			ETS3			ETS3
				ETS3 plug-in		ETS3 plug-in
				SiRoute Installer		SiRoute Inst.
		Archit. Chooser				

Table 5.1 Development activities and corresponding tools

5.4 Adaptation of the FXL framework

In this part the FXL Project is briefly described. Its three main parts: FXL framework; Service Language Layer; and XL platform are presented. Next, different possibilities of adopting the project into a smart home environment, and especially into the Siemens Smart Home, are discussed. Making use of the FXL framework in a smart home environment allows the employment of its concepts and methods for service development.

5.4.1 FXL Project

The FXL project [51] defines the environment for building and running software systems based on Service Oriented Architectures. This includes a platform, similar to OSGi, where services written in different languages can run. Moreover, for the purpose of the project various tools, like (e.g. transformation pipeline builder, transformation libraries), are developed. Since Siemens Smart Home also assumes SOA architecture, the FXL project can deliver some ideas and tools for developing services. Furthermore, the project is cooperation between ETH Zurich and Siemens AG Corporate Technology SE2 (the same department where smart home prototype is developed).

The Flexible XML-based Language (FXL) Project tries to confront various technical difficulties (e.g. complexity, distribution, integration) and business forces (e.g. development time, maintenance costs). The project is focused on Service Oriented Architectures and Web Services.
The FXL project consists of three main parts:

- FXL framework,
- Service Language Layer (SLL),
- XL Platform.

5.4.2 FXL Framework

XML – based Approach (xApproach)
For the FXL project purpose the term xApproach was created [42]. It defines two basic principles: the representation principle and the transformation principle.

The first states that every document written in a non – XML based programming language can be transformed into an XML based representation/model (see Figure 5.4). In other words, the transformation Ta specifies the conversion between the XML model and related views. The view can be traditional, e.g. plain text representation, or derived, e.g. hierarchical or graphical representation (e.g. transformation of xSLL language to human readable form – SLL language).

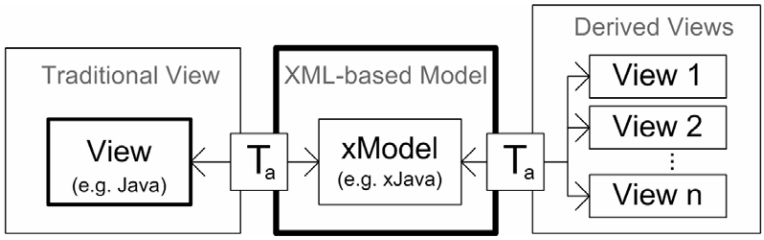


Figure 5.4: Representation Principle

The picture presents the representation principle. Transformation Ta transforms data between different programming languages.

The Transformation principle (see figure 5.5) assumes two additional transformations: for modification of the model (Tb – intra model transformation) and for conversions between models (Tc – inter model transformation).

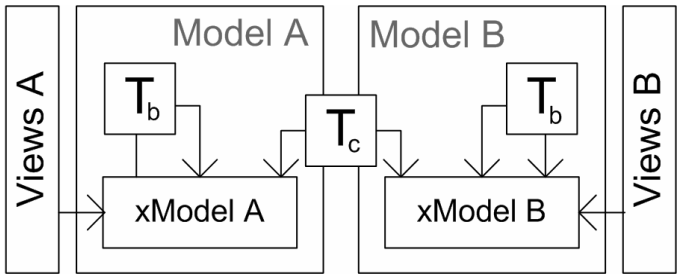


Figure 5.5: Transformation Principle

The picture presents the transformation principle defined in FXL project. Different transformations are possible: Tb – intra model transformation and Tc - inter model transformation.

To illustrate these two principles, assume an application written in xSLL language. Using a Representation Principle (Ta) one can transform the xSLL model into SLL view. The SLL view is more human readable than its xSLL representation. Applying an intra model transformation (Tb), it is possible to change a model internally, e.g. by adding tracing code. Using the last transformation, inter model transformation (Tc), it is possible to represent an application in any other model, e.g. transform xSLL to xJava.

The FXL Core Framework

The main principles of xApproach are implemented in the FXL Core Framework. The processes of transformation between model and views can be built as a workflow graph or XML pipelines. Core transformation is described in Transformation Language (TL). The language provides a higher level abstraction and can be further converted to current XML transformation standards (e.g. XSLT, XQuery).

5.4.3 Service Language Layer

The FXL project introduces a Service Language Layer [25] which gives a high-level abstraction of a service-oriented program (figure 5.6). The main purpose of SSL is to decouple WS programming model (Java, C#, etc) from the execution platform. In this layer a WS is defined in an XML – based intermediary language (xSLL), which is defined according to the main principles of xApproach. From the xSLL representation the program might be transformed for deployment onto one of the available platforms (e.g. Java VM, XL VM, a BPEL engine, etc.).

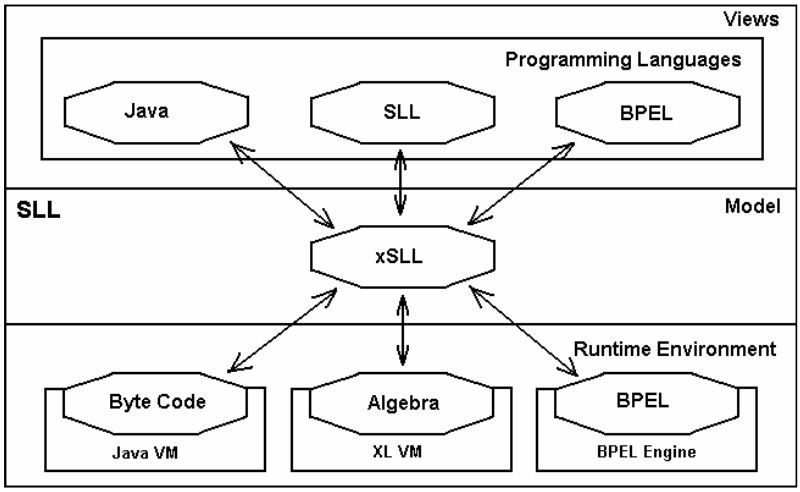


Figure 5.6: The Service Language Layer

Different layer (Views, Model, and Runtime Environment) and its example representations are shown.

The xSLL - based model is easy to process for machines, but difficult to read and manipulate for human beings. Therefore a transformation to SLL language is possible, which is a human readable (java like) view of the xSLL model and is especially useful for writing SOA and WS applications. Transformations between models are possible in both directions, which also gives a programmer the possibility to transform the code between different programming languages.

5.4.4 XL Platform

Applications written in xSSL language can be run inside the XL Platform (xSLL Virtual Machine). The platform exposes application interfaces as Web Services.

The overview of the FXL project components is depicted on Figure 5.7. The FXL framework implements xApproach principles and the XML pipeline mechanism. A specially designed SLL language can be used for service implementation. The final service application runs inside the XL Platform.

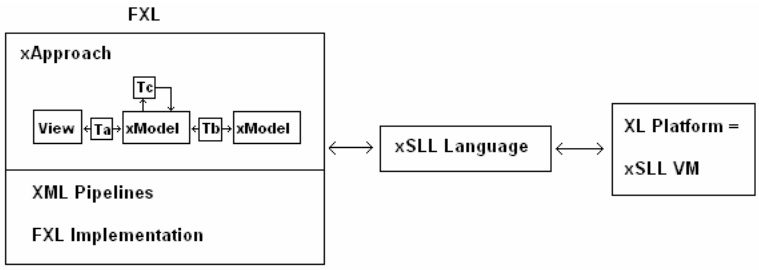


Figure 5.7: Components Overview

Overview of the FXL project components and their mutual relations.

5.4.5 FXL vs. Smart Home

The FXL Project offers an environment (FXL core framework, XL Virtual Machine, tools, Service Layer Language, etc) for development and hosting Web Services. In order to use the FXL project components for smart home purposes, the smart home solution must be based on SOA architecture.

One of the main elements of the FXL framework is the XL Virtual Machine. The XL VM provides a runtime execution environment for applications described in xSLL language. All applications running in the XL VM are in the form of Web Services, i.e. they expose an API to other applications via Web Services. In smart home architecture a proper place to run the Virtual Machine is on the gateway in the home.

Taking into account the recent popularity of WS and associated technologies, applying these technologies for smart home solution has some advantages. Firstly, it is very likely that many services of 3rd parties will be offered in the form of WS. Those services might be then easily integrated within a smart home. Moreover, a special design of the SLL language makes the use of WS very easy and convenient. Secondly, many tools available now for WS and related technologies can be used to develop services.

Developers of smart home applications can also profit from xApproach. The Representation Principle allows the creation of different views from one XML – based model. Those views can support various abstraction levels to support various roles (programmer, operator, end user, etc.) and phases (development, operation, maintenance). In can increase the understanding and communication between various entities involved in a development process.
The Transformation Principle allows a modification of the base XML-model (insertion or removal of block codes, whole classes and files). In this way agility aspects as well as language and software personalization/customization problems, are supported.
Furthermore, transformation between base models gives the ability to generate code for different hardware and software components (e.g. backend - EJB, gateway - OSGi, PDAs – C#, controlled devices, etc.). It can be done by applying different transformations and a XML pipeline mechanism to one, XML – based, service model.

5.4.6 FXL vs. Siemens Smart Home

Siemens Smart Home already has a working service – oriented solution. It integrates various devices at home and offers services for inhabitants. There is also specification of the Siemens Smart Home describing all software and hardware components, protocols and services. To apply FXL project for the Siemens Smart home one has to make some changes to already developed and specified architecture. The scale of those changes depends on how the FXL framework is integrated with the Siemens Smart Home.

Integration with OSGi framework
It is possible to integrate the FXL framework without many changes to the existing Siemens Smart Home solution. The XL platform can run on the gateway inside the OSGi framework. In this case there is no need to change the implementation of old services, since the SLL language enables an easy method to integrate java programs. New software can be written with the help of the FXL framework (FXL tools, SLL, etc.), which gives all the advantages described above.

However, this solution has some drawbacks. Firstly, two platforms coexist together: the old OSGi platform and the new FXL platform. This, and the fact that the FXL engine runs inside an OSGi platform might cause some performance problems. The gateway device at home has a relatively small computation power. Another problem is the time and effort needed to learn a new language (SLL) and XL platform. It would be also necessary to write new Siemens Smart Home specifications and documentations completed by a new software model.

The management platform for controlling the FXL platform from remote (from Backend server) is also not available. The need to develop such a platform might be also time and money consuming.

The advantage is that a developer has the possibility to use the whole FXL environment. First, new services can be programmed with Service Layer Language, which is especially convenient for service development and composition. Second, the service may be written in any other language (e.g. BPEL, Java) and than transformed using FXL libraries to Service Layer Language. During the development all the tools created for the project purpose are available for the developer [51]:

- ANTLR (Another Tool for Language Recognition) grammar integration
- basic Java project support (compilation, execution, xBuild, etc.)
- xViews (Lang, xLang), Transformation Libraries (including predefined XLST, Merging, etc. processes)
- visual XML Pipeline Editor (modification, execution, etc.)
- SLL1.1 and TL1.0 support

Substitution of OSGi framework

In this case, the OSGi framework is totally substituted by FXL framework. The advantages and disadvantages of this solution have already been described. In contrast to previous solutions, there will be no problem with two coexisting platforms. On the other hand all software and already implemented services would have to be rewritten for the FXL platform.

6 Proposed tool chain for Service Development

In this chapter the service implementation process for the Siemens Smart Home is investigated more deeply. Next some proposal improvement and tools are presented. Similarly, in the next section the testing method in the Siemens Smart Home and corresponding tools are described. At the end the idea of Service Development Support System is presented.

6.1 Introduction

The first four stages of service development, i.e. requirements elicitation, analysis, service design and object design are similar to the general software development processes described in literature [4]. Similarly, tools like UML diagrams or CRC cards [13] used in those processes have no domain specific features. Therefore, there is no need to develop any special method or tools for these procedures.

For version and configuration management there are already good tools (please refer to the previous chapter), so the development of a new, powerful application would be a waste of resources.

The deployment of services in the Siemens Smart Home is to a large extent an automatic process. First, all service files (OSGi bundles, descriptor files, etc) must be uploaded to the backend server (SEP) via a Web Service interface. Next, a user needs to subscribe to a new feature. After that, all the required applications are deployed on the home gateway (SGP) and other devices (e.g. PDA) via a management link. Since the deployment is already almost fully automatic, there is no need to develop any special deployment tool. The situation is a bit different when a new service enforces an installation of some devices at home (e.g. EIB or SiRoute system). In this case a technician for installation and configuration of devices is needed. However, this process cannot be easily automated by a software tool, since e.g. installation of electrical devices (EIB devices) needs to be done by somebody familiar with the security guidelines.

Looking closer at the implementation and testing, there are still some parts which can be improved and systematised. Concerning the implementation, a lot of code can be generated from a new service XML description file. In the case of testing, some mock-up components can be automatically created to enable easier unit testing. Also in a distributed, heterogeneous environment a good testing method is important. These issues will be covered further in this chapter.

6.2 Implementation

In this section the current status of the implementation activity are depicted. Next, some tools for the improvement of this process will be proposed.

6.2.1 Current status

The figure 6.1 presents various activities during the implementation of a smart home service. In this case the service runs locally on the home gateway. Most of the activities are performed by developers supported by some general tools, e.g. Eclipse, Visual Studio.

The first step is to write an UPnP service interface (process 1 on the figure 6.1). The format of the service interface (UPnP) is forced by the architecture of the Siemens Smart Home, where the UPnP is an integration level for home devices and most of the services. If the service one wants to im-

plement is already standardised by the UPnP Forum, the documents can be downloaded from the UPnP web page (www.upnp.org). However some adjustments might be necessary, e.g. when the developed service provides some additional features in comparison to the standardised service. For example, the descriptions are available for digital security camera and lighting controls. If the service is not standardised yet, the description must be completely created from scratch. This specification serves as an entry data for all other implemented service parts. The description, in an XML format, should also be translated to Java interface format for further development. The UPnP Java interface defines method templates, which correspond to the XML UPnP description.

Based on the UPnP Java interface various components which reside on the gateway are implemented: the main service implementation, UI Service Face and SEP – SGP Communication (1.6.1, 1.6.2, 1.6.3 on the diagram). The service implementation component is the core service logic and is deployed on the Service Gateway Platform at home. Since it runs inside the OSGi platform, the application must be written as an OSGi bundle.
The UI Service Face component is the part of the Web User Interface Extensibility Platform. According to the UIEP architecture a service face defines a UI but is independent of the different rendering devices. It is possible by introducing of UI Rendering Framework of each RC device. Each framework exposes the same UI Service API. A UI Service Face is a mediator between a service API and the UI Service API, thus it contains service specific as well as UI specific parts. Each service has its own UI Service Face.
The SEP – SGP Communication component is only present when a service or part of it is implemented on the backend server. It is responsible for proper communication between the SEP and SGP via Web Services.
The Device –IP Proxy component is used when new specific devices have to be integrated. Some appliances might have an IP address for remote control (e.g. an internet camera controlled from outside the home). At the same time the control should be made without the mediation of the Smart Home API. Since in the Siemens Smart Home those devices are behind the Network Address Translation protocol (NAT), they might be not accessible from outside. The task of the Device –IP proxy component is to enable this communication. The communication and component is independent of the UPnP interface.

Together with the development of service logic a number of description files should be written by a programmer (2.2 – 2.5 on the figure 6.1). Those files consist of various data like:

- Service details (e.g. name, URL for service front pages),
- Service description (e.g. Auto Configuration Server scripts),
- Metadata required for building subscription pages on portals,
- Metadata specifying extra parameters.

The next step is the integration of home devices (2.1-1 on the diagram). This is only done if the devices have not been integrated before and they do not support UPnP protocol. To integrate the appliances with the Siemens Smart Home a bridge must be written. The proxy software makes the devices available as UPnP devices by performing the transformation between a device-vendor proprietary protocol and UPnP protocol. The bridge can be written in any programming provided the home gateway supports it. Since the OSGi framework is deployed at the SGP there is good reason to write the bridge as an OSGi bundle (assuming the device has IP connectivity).

For the PDA a Rich-client UIEP has been developed. The UI for the new service has to be provided as an UI plug-in component written in C#. On the PDA the UI has access to service API (available normally on the gateway) via a service proxy implementation. Those two steps are illustrated on the diagram as steps 2.7.1 and 2.7.2.

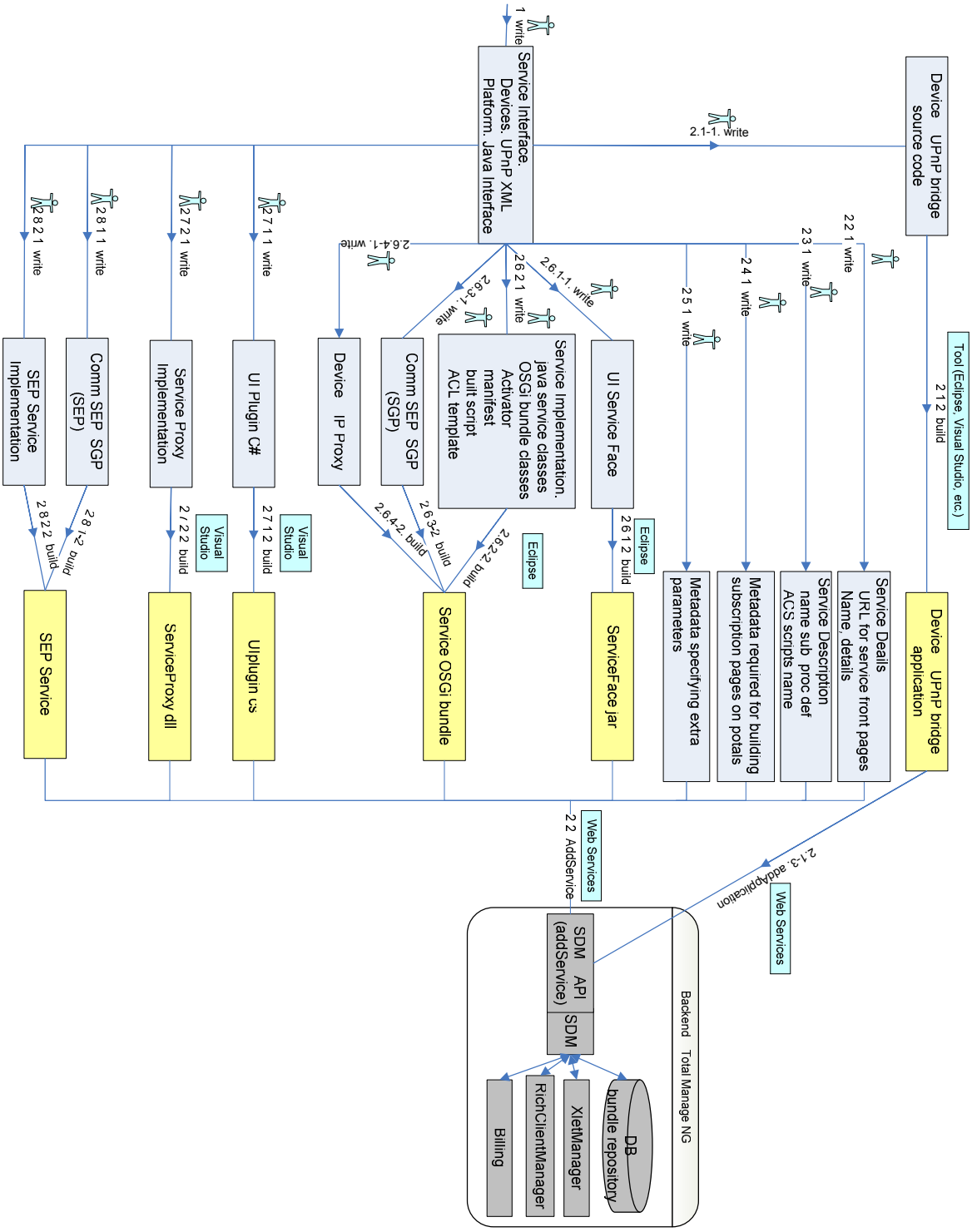


Figure 6.1: Current service development process

The diagram depicts the development of a service running only at the home gateway. Developer activities, used tools and developed components are presented.
Legend: arrows – development processes; number above arrows determine the sequence of the processes, e.g. 1 – first activity; after that the processes 2.1-1, 2.2.-1 2.6.1-1 (…) can be performed in parallel; after the process 2.1-1 the activity 2.1-2 has to be carried out. Grey boxes above arrows – used tools; human symbol – activity performed by a developer; light grey boxes – components implemented during the process; dark grey boxes – components in the backend system

When all the service components are written and compiled (description files, OSGi bundles, UI plug-in, etc.) they can be uploaded to the Service Enabling Platform. The backend system has a Service Deployment Manager component (SDM) which exposes its functionality via API. The upload of service components is done via web services (using the SDM API).

The process described above presents the development of a service completely running at home. In general there are more types of services available in the Siemens Smart Home (for the full list please refer to chapter 3.4.2):

- Those that make use of information in the Internet (e.g. Information and Web-Based services)
- Those that are also deployed at the backend (e.g. video on demand, health service).

Information and Web-Based services are relatively simple to implement, and thus they are less interesting. Because these services are already implemented somewhere and are available on the Internet, a developer only has to program various UI for the Siemens Smart Home system. There is no need to implement service logic (OSGi bundle) or a Device-UPnP Bridge. For these service types there is no need to create an UPnP description. Instead it is very probable that a document in Web Service Description Language format (WSDL) will be provided. It can serve as an input for some skeleton code generation (as UPnP description for local services).

For the services which are also deployed at the backend an additional service component running on the SGP has to be written. However, this case is not considered here in detail. By the time of writing the Service Enabling Platform was not fully implemented.

6.2.2 Additional tools

During the development process depicted on the previous diagram, the programmer is supported by some general programming tools (e.g. IDEs). Among many IDEs for java development Eclipse was selected. The benefits of this choice are an open source character of Eclipse, broad popularity in industry, good plug-in architecture and many available plug-ins on the market. The C# development is made in Microsoft Visual Studio, since it provides a good support for development of .NET CF applications. Expanding this simple set of tools by some additional applications can bring some benefits in the development process. Especially plug-in architecture of Eclipse and Visual Studio makes it easy to extend these IDEs by some specific plug-ins.

At first a number of generators should be implemented. Based on the UPnP device or service description file they can automatically create source code and skeletons for different components. This saves time, since developers can focus on writing component logic. They are also more secured from making errors while programming a general part of a component.

Developers have to write in many places description documents, which are often in a format hard readable by a human (e.g. UPnP device description in XML). By introducing some tools with a proper GUI this process can be much easier and faster.

The programmers at some point in the process have to make a decision concerning further development (e.g. what architecture to choose for an implementing service). These decisions can be made with a tool support. In this case programmers do not need know all sophistication details about smart home domain.

The development process supported by a number of additional tools is illustrated at the figure 6.2. Next, the proposed tools are explained.

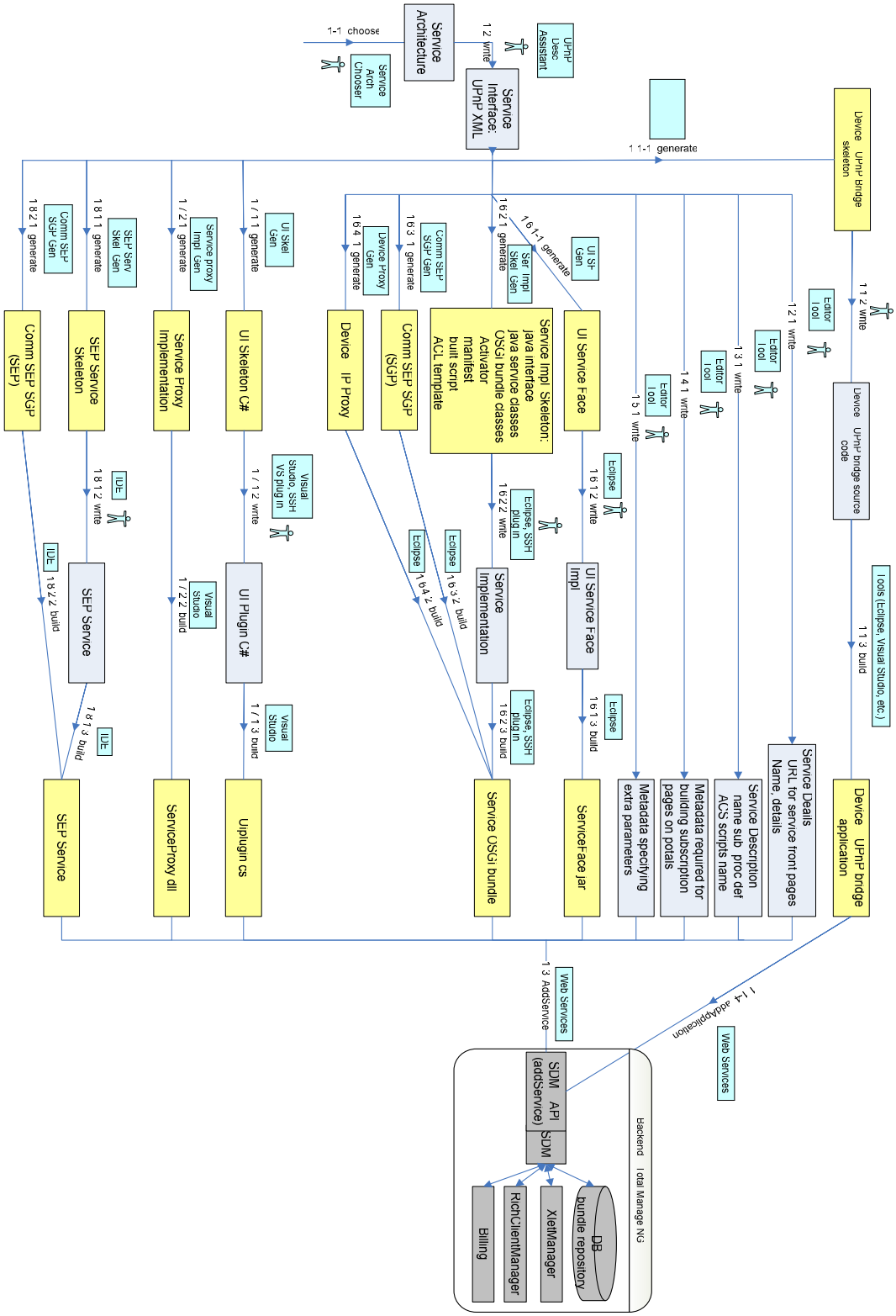


Figure 6.2: Development process with tools

The diagram depicts the development of a service running only at the home gateway. The diagram is extended by some additional development tools.
Legend: arrows – development processes; number above arrows determine the sequence of the processes, e.g. 1 – first activity; after that the processes 2.1-1, 2.2.-1 2.6.1-1 (…) can be performed in parallel; after the process 2.1-1 the activity 2.1-2 has to be carried out. Grey boxes above arrows – used tools; human symbol – activity performed by a developer; light grey boxes – components implemented during the process; dark grey boxes – components in the backend system

6.2.2.1 Service Architecture Chooser

At the beginning of the service development a programmer must make decisions about the architecture for implemented service (process 1-1 on the diagram). Depending on the service type and its architecture different components have to be implemented. A programmer relatively new in the smart home domain might be confused by this decision. As a result the decision can take too much time and might be wrong. A good application supporting this action is a tool developed by J. Wechsler (see chapter 5.2.6). A developer is asked a set of simple questions about the implementing service. They do not require any sophisticated knowledge about the Siemens Smart Home. Based on the answers the most suitable architecture is proposed.

6.2.2.2 Service Architecture Construction tool

Naturally the architecture proposed by the Service Architecture Chooser is not binding and a developer is free to make any changes. The Service Architecture Construction is a tool which aims to assist programmers during designing/redefining the service architecture. Using GUI he is able to add/remove various Siemens Smart Home specific components and connect them together. A more detailed description of this tool is in chapter 6.5.

6.2.2.3 XML UPnP Service Interface Description Assistant

Having defined service architecture a programmer must create an UPnP XML service description document (process 1-1 on the diagram). Since the XML format is not easy readable for humans, it would be good to have a GUI tool for this task. Such a program is available from Intel. Service Author application is available in the package Intel Tools for UPnP Technologies (described in chapter 5.2.1). It provides a convenient way to create UPnP services (adding new action, defining parameters, etc.). After a developer uses GUI to insert or/and modify data, an UPnP XML description file is automatically generated. However, the Service Author has one drawback. It can not create an UPnP device. Therefore, a new application for that purpose should be written or the Intel tool should be extended.

6.2.2.4 Device – UPnP Bridge Skeleton Generator

The next tool is a Device – UPnP Bridge Skeleton Generator (process 1.1-1 on the diagram). The tool makes use of the UPnP description file and generates bridge skeleton files. The bridge maps a device specific protocol to the UPnP protocol. In general, most of the UPnP part of this application is standard for all the devices. Therefore, a general architecture of the bridge can be easily defined and generated for each UPnP description file. The skeleton must be than filled with application logic specific for each proprietary device protocol and corresponding UPnP device.

6.2.2.5 Various editor tools

With each service there can be various configuration files connected: service details, service description, metadata required for building subscription pages on portals, metadata specifying extra parameters (processes 1.2 – 1.5 on the diagram). They are created in an XML format, which is not easily readable by a human. An editor tool can support a developer to create such configuration files. Firstly, it reads the UPnP description and based on that creates some default data. Secondly, a programmer can easily change the configuration data using GUI provided by the tool. The application can be modelled on a tool delivered with Eclipse. The tool is for editing OSGi bundle configuration metadata when developing an Eclipse plug-in (see figure 6.3).

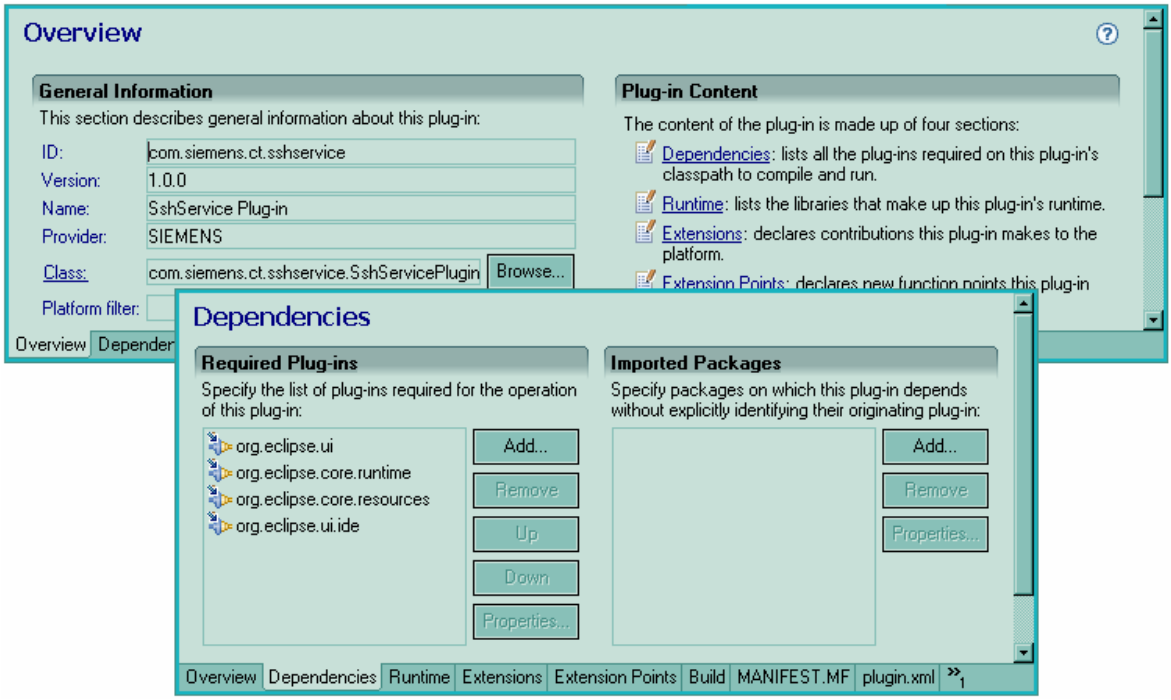


Figure 6.3: Eclipse tool for OSGi bundle metadata

The screenshots present an Eclipse tool for editing Eclipse plug-in metadata. Since Eclipse is based on OSGi the tool presents also metadata for an OSGi bundle.

6.2.2.6 UI Service Face Skeleton Generator

A UI Service Face is a mediator between a service API and the UI Service API. The UI service API defined in the UIEP is constant and agnostic to the implemented service. The service API depends on the service and is fully described in the UPnP description file. As a result a lot of UI Service Face code is general. The UI Service Face skeleton can be generated from UI Service API and UPnP description. The skeleton has to be then filled with some specific UI design logic by a programmer.

6.2.2.7 Service Implementation Skeleton Generator + OSGi Bundle Creator

The most important service component encapsulating application logic is the service OSGi bundle running on the Smart Home gateway. It is responsible for providing an UPnP interface for use by the other components in the system. Therefore, much of the code can be generated from a UPnP description file. Moreover, since it is an OSGi bundle, the component must follow the OSGi specification. Again some OSGi specific files and code can be automatically created, e.g. OSGi bundle activator, classes, and manifest.

6.2.2.8 Siemens Smart Home plug-in

The service OSGi bundle skeleton must be filled with the application logic as well as with OSGi bundle specific code (process 1.6.2-2,3 on the diagram). This process can be supported by a special Smart Home Gateway plug-in. The plug-in extends Eclipse (or other IDE tool) functionality by adding some special views, windows, Siemens Smart Home auto competition code, etc.

A new tool should be written to support writing Siemens Smart Home specific code. In the case of OSGi specific code there are already some plug-ins available on the market: the Knopflerfish Eclipse Plug-in (free) and the OSGi ProSyst Eclipse Plug-in (commercial). The can also build the project and automatically create OSGi bundle files, which are ready to direct deploy in OSGi platform. For more information about these plug-ins please refer to chapter 5.2.2.

The plug-in should also integrate other components, e.g. SEP – SGP Communication component (in the case when part of the service runs on the backend) and Device – IP Proxy (when a device with an IP address has to be accessible from outside).

6.2.2.9 Communication SEP – SGP

When a smart home service is fully or partly deployed on the backend server, there must be two components (one on the SGP and one on the SEP) which handle the whole communication burden. Since Web Services are used for communication, both components can be entirely generated from Web Services description document (processes 1.6.3-1 and 1.8.2-1 on the diagram).

6.2.2.10 Device Proxy Generator

Some smart home devices can have IP/TCP protocol built in for external communication. However, due to NAT protocol the communication from the outside world is impossible. The Device – IP Proxy component should help to overcome this problem. The component can be fully generated by a Device Proxy Generator.

6.2.2.11 UI Skeleton Generator

Almost all of the services should be available via a Personal Digital Assistant (PDA). A relatively good display, small size and wireless capabilities cause a PDA a good human interaction device. For the PDA a User Interface Extensibility Platform (UIEP) concept defines rich client UI architecture. Various service plug-ins, written in C# language in .NET CF environment can be easily downloaded and installed on a PDA. Because of the standardised rich client architecture and the fact that a UI should enable inhabitants to use UPnP services, much of the code is generated (process 1.7.1-1 on the diagram). UI architecture specification and UPnP description file will serve as an input.

6.2.2.13 Siemens Smart Home Visual Studio Plug-in

Siemens Smart Home Visual Studio Plug-in supports the development of a Service PDA UI. Similar to the Eclipse plug-in it adds Siemens Smart Home specific views, windows, auto competition code mechanism, etc. Additionally, the tool should provide a convenient interface to create Smart Home Service PDA GUI (Siemens Smart Home specific widgets, colour schemas, etc.).

6.2.2.14 Service Proxy Implementation Generator

The UI software on a PDA listens to the events fired by a user and maps these events to the Siemens Smart Home service API calls. Since the API is available on the Service Gateway Platform, a PDA uses Wireless LAN protocol and Web Services to fire corresponding actions remotely. For this purpose the plug-in uses a Service Proxy component which exposes service API available on the SGP. This makes the whole process transparent to the plug-in. Service Proxy component maps UPnP action calls to the UIEP specific functions. The UIEP intern components fires corresponding actions on the SGP platform. Since there is no logic in the Service Proxy component, it can be completely generated by a tool (process 1.7.2-1 on the diagram). The UPnP description file and UIEP specification will serve as an input for this process.

6.2.2.15 SEP Service Skeleton Generator

In the case of a smart home service running fully or partly on the backend server, a SEP Service component must be created. It runs on the backed and implements the service logic. From the UPnP service description file a SEP Service Skeleton can be generated.
More detailed information about the integration of services with the backed are unfortunately unavailable. At the time this work was written the SEP platform was not fully designed.

6.2.3 Component deployment architecture

The development process of service components is shown in the previous chapter. These components are than deployed on hardware devices in the Siemens Smart Home (see figure 6.4). The main hardware part is the Home Gateway with the SGP running in OSGi Framework. New service files (i.e. ServiceFace.jar and serviceBundle.jar) are running on the SGP. Additionally, when home devices do not support the UPnP protocol, the home gateway will host any Device – UPnP bridge

application. If the UPnP protocol is supported by home appliance, the devices can be directly connected to the gateway. The gateway is also connected via the Internet with the Service Enabling Platform, which resides on the operator side. Apart from management and monitoring tasks the backend platform can host a service part (for service which also runs on the backend). The rich-client UIEP platform runs on the PDA and thus a UI plug-in and Service Proxy components are deployed there. Inhabitants can also use the smart home services via a browser and thin-client interface. A PC must be than connected via the Ethernet with the gateway.

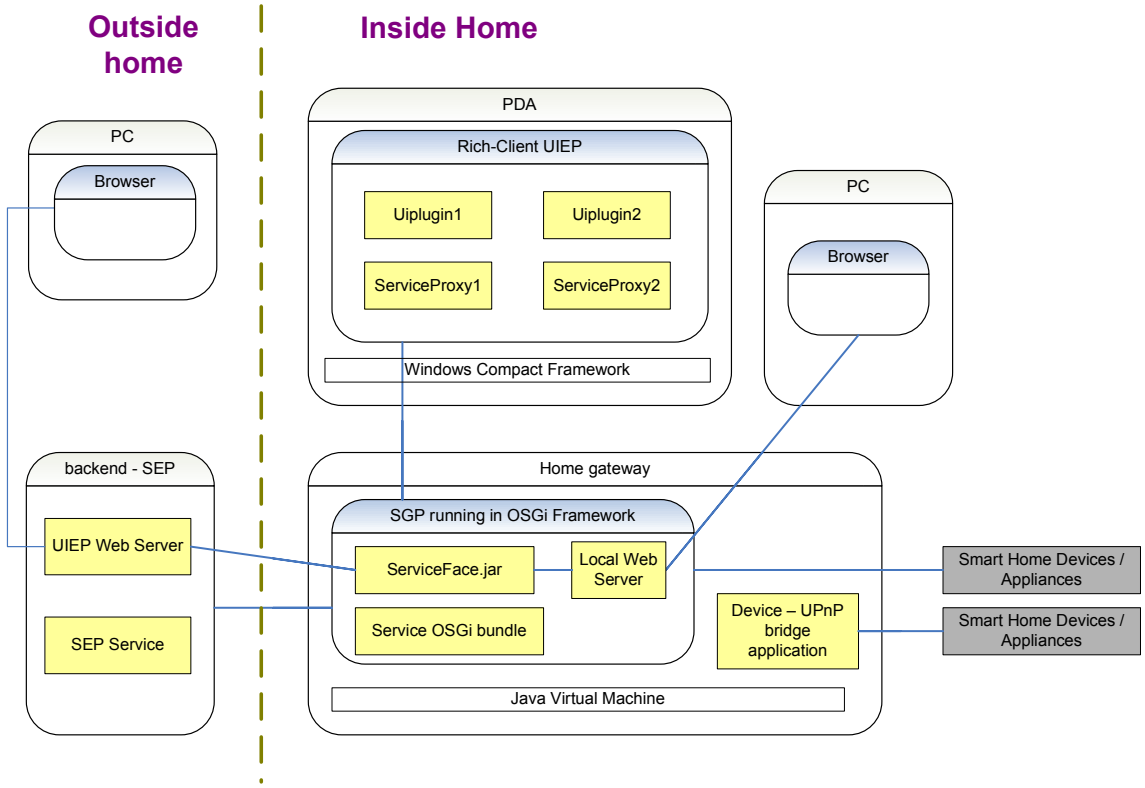


Figure 6.4: Deployment of service components

The picture presents deployment of service component in Siemens Smart Home architecture.
Legend: dark grey boxes – real appliances at home; light grey boxes – software components; exterior frames – hardware components; interior frames – software platforms/frameworks.

6.3 Testing

In this section, the testing process in the Siemens Smart Home is explained. The various components needed only for testing are introduced. Next, a process for implementing those components is introduced.

6.3.1 Test use cases

In heterogeneous and distributed environments, such as Siemens Smart Home, the testing is especially difficult. At first all the developed components need to be evaluated in separation. Then the integration tests between two or more software parts have to be performed. This is accomplished by testing all the components together. The next step is to carry out system and usability tests. The first aims at answering the question of whether the developed service meets functional and per-

formance requirements. The latter is performed together with users in order to measure the level of their satisfaction using the new service. For the heterogeneous, distributed system it is also important to have a good method for remote debugging and reporting.

Testing processes in the Siemens Smart Home are depicted on figure 6.5 and 6.6. Firstly, during unit tests a number of service components are tested (see figure 6.8):

- Service Face (Web UIEP),
- UPnP Bridge (integration of legacy devices),
- OSGi bundle (service logic running on the gateway),
- PDA UI plug-in (rich client PDA UIEP).

The next step is to perform the integration tests. This process can vary in different services and depends on the type of service implemented, number of proprietary devices controlled and existence of additional service specific components. In general, service integration tests include the examination of interaction between components (see figure 6.8):

- UI plug-in on the PDA with OSGi service bundle on the Service Gateway Platform
- Web user interface (Service Face) with OSGi service bundle
- UPnP Bridge and proprietary devices with OSGi service bundle
- All components together (UI plug-in, Web UI, UPnP Bridge with proprietary devices, OSGi service bundle)

During unit and integration tests various tools (introduced below) might be useful. Their main task is to allow for testing various components in isolation.

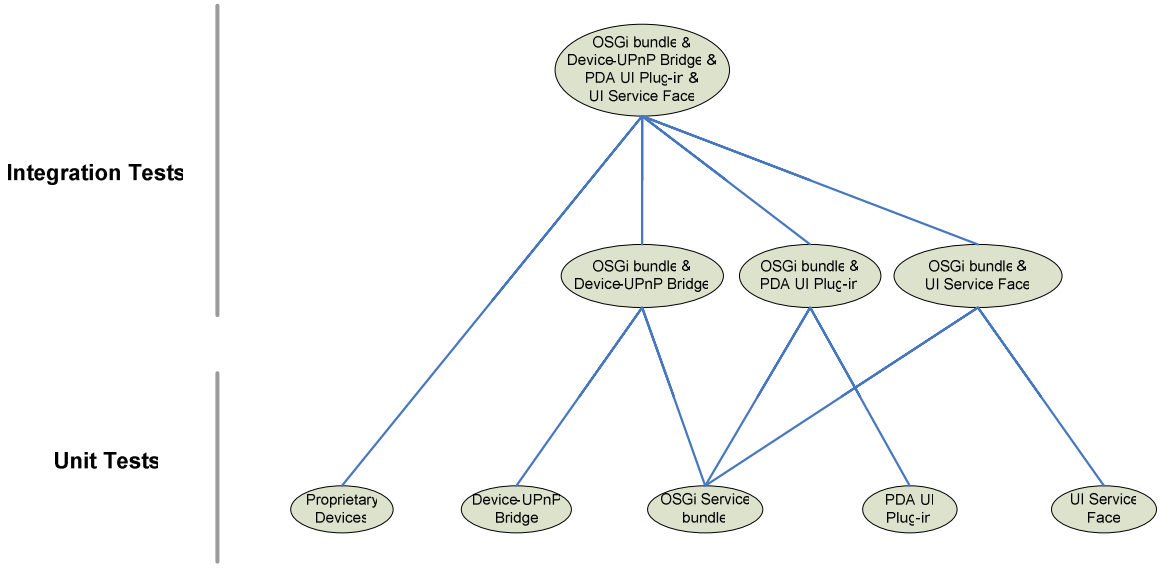


Figure 6.5: Testing Siemens Smart Home service components

The diagram depicts test activities for Siemens Smart Home service components. At the lower level unit tests are performed. The middle and higher level presents integration tests.

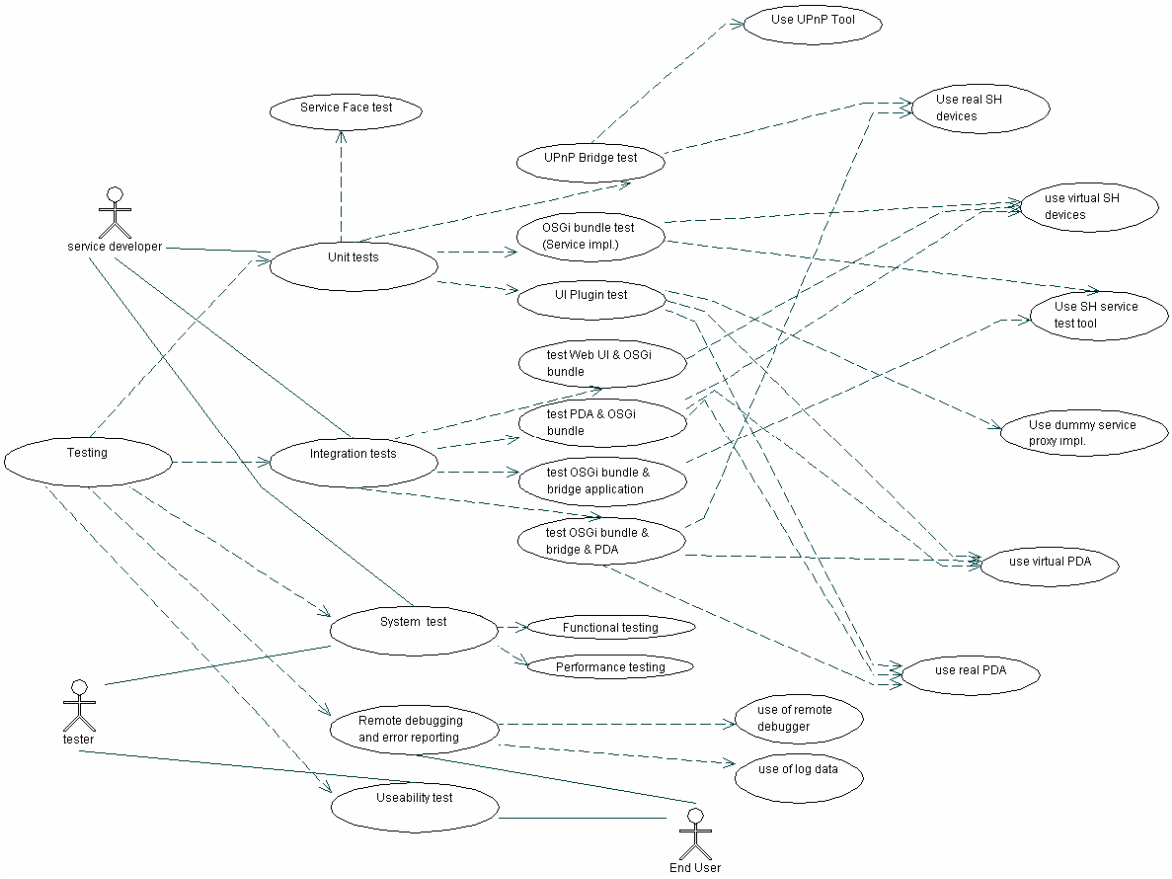


Figure 6.6: Testing in Siemens Smart Home – use case diagram

The use case diagram presents testing and debugging activity of a service in the Siemens Smart Home.
Legend: dashed line - <include> stereotype

Due to the distributed and heterogeneous nature of smart home environment, remote debugging and error reporting issues are also quite important. First, a developer must have the possibility to debug a service bundle, running on the SGP, from his computer. Next, some bugs might come out when the system is already installed at the customer. The use logging system might be crucial in this case. The tools which are helpful during testing and debugging a smart home service are described below.

6.3.1.1 UPnP tools

Both UPnP tool sets described in chapter 5.2.1 include an UPnP generic control point. This enables the control of each UPnP devices via firing action of a device or service. The developer can simply test if the device works correctly at the UPnP protocol level and if particular UPnP actions are well implemented.

Additionally, an Intel package includes a Device Scriptor tool. It allows a developer to quickly create UPnP scripts that run against a set of target UPnP services. For example, Device Scriptor can be used to test various scenarios on a UPnP device, or test interaction between multiple UPnP devices.

6.3.1.2 Virtual Device (new)

The Virtual Devices might be helpful while testing the type of services which make use of real proprietary devices. For example, to test lighting control without a real device a simple virtual light application running on the computer can be implemented. A developer can use buttons in the application to switch on and off the virtual light (and thus become status events like working with a real

light). Similarly, he will see the light switched on or off while firing UPnP actions. The Virtual Device has to be developed for each type of device separately.

6.3.1.3 Smart Home Service API test tool (new)

"New" means that the tool is not available on the market. The Smart Home Service API test tool should enable the direct checking of Siemens Smart Home API. Usually the service is integrated into Siemens Smart Home architecture and the service can only be accessed using UI and Web Services as a remote access protocol. This tool resides on a computer connected to the Siemens Smart Home network and allows the testing of the service functionality without UI mediation. Various Web Services actions are invoked on the gateway and responses can be check for correctness. Some SOAP and Web Services tool suitable for this task were described briefly in chapter 5.1.9.

6.3.1.4 PDA simulator

A PDA simulator enables the testing of a developed UI plug-in before installing it on the real PDA. Thanks to the tool simple bugs and errors can be easily removed. The PDA simulator is available together with Microsoft Visual Studio IDE (see chapter 5.1.3).

6.3.1.5 Dummy Service Proxy Implementation (new)

A Service Proxy Implementation is deployed on a PDA and allows UI plug-ins to use a service API. It serves a mediator between service bundles running on the home gateway and UI plug-in. A Dummy Service Proxy Implementation has no connection to the home gateway. It allows testing UI plug-ins in isolation, i.e. without calling the corresponding service bundle. The Dummy Service must be implemented for each service separately.

6.3.1.6 Remote debugger

A Remote debugger is needed to test various service components when they are deployed on the corresponding devices (e.g. test service bundle running on SGP, UI plug-in deployed on the PDA). Both Eclipse and Microsoft Visual Studio deliver remote debugger functionality.

6.3.1.7 Logging framework

Due to the distributed character of the Siemens Smart Home solution, many bugs can appear later, when service components have already been deployed. To cope with this problem a good logging mechanism can be used. The main features a logging framework should provide are configuration at runtime and sending log data to a remote server. The first property allows, for example, to change the log level, e.g. DEBUG, ERROR. Debug mode is on one hand useful when an error needs to be localized; on the other hand it uses more computation, bandwidth and storage resources. Error mode on the contrary uses fewer system resources, but gives only some general information. For description of some useful logging frameworks please refer to chapter 5.1.8.

6.3.1.8 Dummy SEP Service Implementation

This is a way to test services which are fully or partially running on the backend. A Dummy SEP Service Implementation is a mock component which has the interface of SEP Service and no service logic. The tool can be deployed on a PC connected to the home gateway (like the SEP platform). This should simulate the backend server with a service part running on it. A developer can fire Siemens Smart Home gateway action and check responses for correctness.

6.3.2 Implementation of test components

Two components described above: Virtual Devices and Dummy Service Proxy Implementation are specific for each service. The development process of these components is depicted on figure 6.7. The first two processes (1-1 and 1-2) are the same as the development of standard service parts (see

figure 6.2). After that, based on the UPnP description and a service type, the test components are implemented. For this purpose three additional tools might be used:

- Virtual Device Skeleton Generator (process 1.8.1-1)
- Dummy Service Proxy Generator (process 1.8.2-1)
- Dummy SEP Service Generator (process 1.8.3-1)

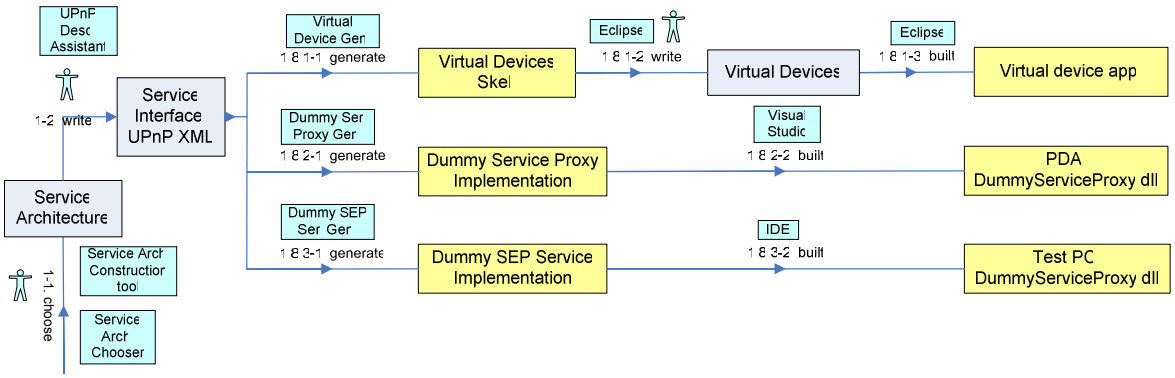


Figure 6.7: Development process of test components

The diagram depicts a development of service components which are used only for testing purposes.

6.3.3 Test components deployment

On the figure 6.8 Siemens Smart Home architecture for testing purposes is presented. Additional to service parts described in chapter 6.2 there are a number of components only for test purposes. The Dummy Service Proxy is deployed on a PDA. On the Service Enabling Platform there is Dummy SEP Service. For development and test purposes two more PCs are needed. On the first one PDA simulator, remote debugger, Web Services test tool and UPnP tools are deployed. On the second Virtual Devices and ETS software for direct testing of EIB devices are running. The reason for testing PCs is to separate an UPnP Control Point (UPnP tools) from UPnP Virtual Devices and thus enable testing service components distributed over various hardware parts.

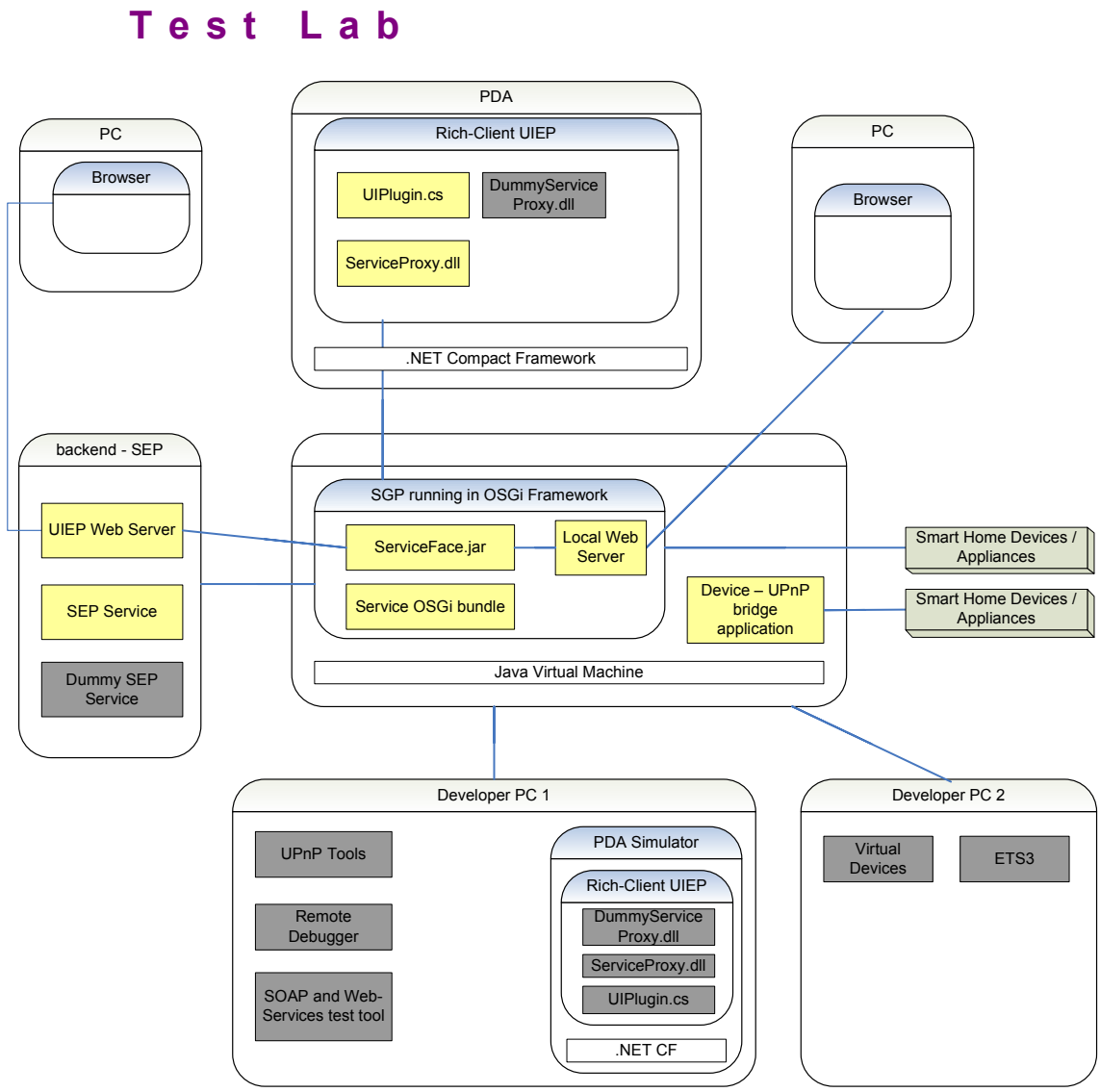


Figure 6.8: Deployment of service components

The picture presents deployment of service component and test components in Siemens Smart Home architecture.
Legend: grey 3d boxes – real appliances at home; light grey boxes – software components; dark grey boxes – tools or software components for testing purposes; exterior frames – hardware components; interior frames – software platforms/frameworks.

6.4 Service Development Support System (SDSS)

Another tool which aims to assist the development of smart home services is Service Development Support System (SDSS), which supports the implementation and testing of a new service.
The main tasks of the SDSS system are:

- Extend IDE functionality so the developer can make use of various features, e.g. Siemens Smart Home specific auto completion code, Siemens Smart Home API browser.
- Support implementation of services which makes use of other, already developed services
- Provide various Siemens Smart Home system components for testing purposes
- Classification of services
- Automatic updates of Siemens Smart Home software

At first the developer should be supported via common IDE features while developing new services. This includes auto completion code when a programmer wants to use a Siemens Smart Home specific function, variable, class, etc. The used Siemens Smart Home software components should be recognized and included in the project class path. The Siemens Smart Home API browser enables to a convenient search for any class/method/variable and useful information about it.

The next task of the SDSS system concerns the development of a service which makes use of other, already developed services. When an existing service was developed by an other company or operator the SDSS allows connection to this company and downloading of all necessary components. However, when there are many operators and developers hosting their own service databases, searching for any service and downloading components might be a problem. The tool should provide a convenient method to find that service which can reside on remote, third party servers. Since all the Siemens Smart Home software is classified in a coherent way, searching for needed components should not be a problem. Further, in order to make use of this service in the implementation, the SDSS system has to download all the necessary service components. This might include, for example, the OSGi service bundle file, UI Service Face jar file, Service UI plug-in for a PDA, etc.

After implementation the service must be well tested. For this purpose a developer has to set a local testing environment. In order to do this the SDSS system enables to download preconfigured platforms (SGP and SEP), all necessary services and additional test tools. Various components stored in the system are depicted on figure 6.9.

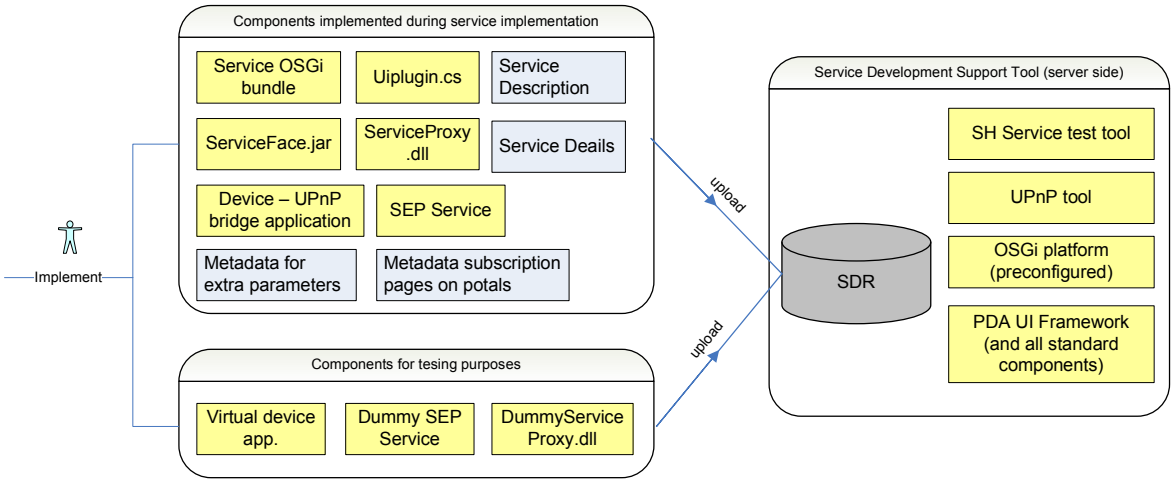


Figure 6.9: Different components stored by SDSS system

The diagram shows various service components and Siemens Smart Home software stored at the Service Development Support System (on the server side).

Another task of the SDSS system is to make automatic updates of Siemens Smart Home software. During testing and development of service some basic platform components can change, e.g. a new version of Service Gateway Platform or a new version of Device – UPnP Bridge.

The service search functionality is used in SDSS system in many places together with other features. Therefore, it is important to set reasonable rules for service classification. The J.Wechsler master thesis [64] can be served as a good starting point. J.Wechsler proposes a basic classification method for smart home services (see chapter 3.4). Of course, this approach should be extended, since the number of available services might be potentially very large.

6.4.1 SDSS Architecture

The SDSS system consists of many components which can reside on different servers (see figure 6.10). The core system components reside on the main server at the operator/service integrator side. It is important to note that there can be more than one service operator. The SDSS service is also located on the developer's server. On each side there are special components, ensuring communication and exchange of data.

The ASDS system consists of three main components at the operator's side:

1. Service Development Repository (SDR)
2. Service Development Download Manager (SDUM)
3. New Service Upload Manager (NSUM)

At the service developer side there are modules:

1. Service Development Download Client (SDDC)
2. Service Development Upload Client (SDUC)
3. Local Service Development Repository (L-SDR)
4. Implementation Manager – Eclipse plug-in (EIM)
5. Implementation Manager – Visual Studio plug-in (VSIM)
6. Implementation Manager – for other IDEs

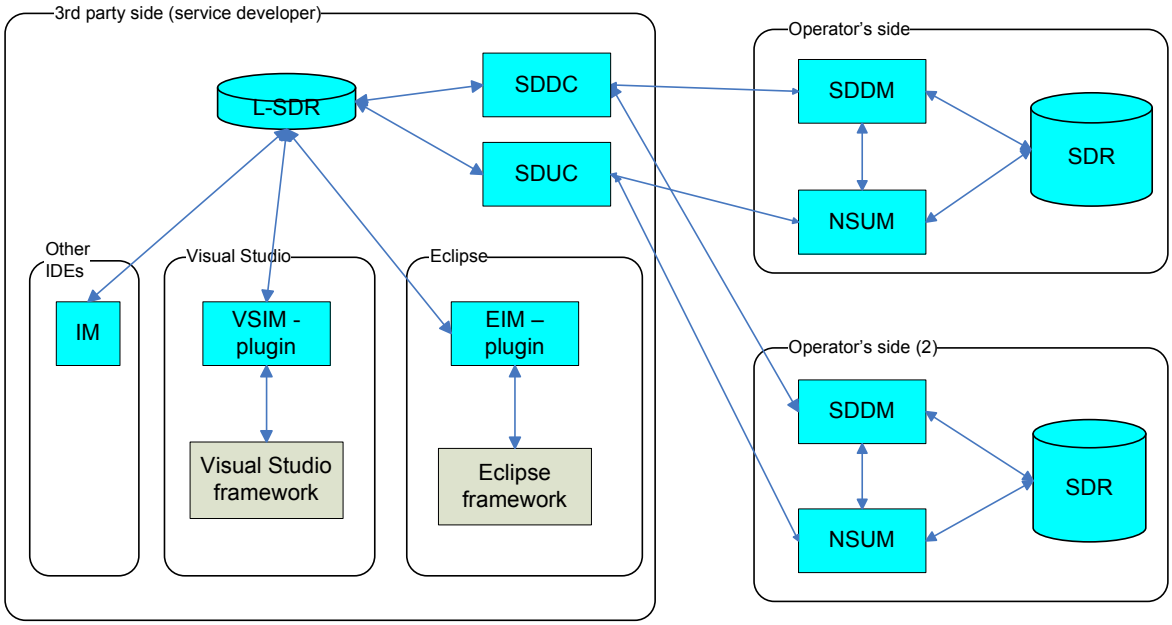


Figure 6.10: SDSS Architecture

The picture depicts the architecture of the Service Development Support System. Legend: dark grey boxes – components of the SDSS system; light grey boxes – other software components (e.g. Eclipse framework).

Service Development Repository (SDR)

The main purpose of the SDR component is to store all data which might be needed while developing services. This includes:

- All services which are available for users at operator's platform; this includes service names, software components (OSGi bundle, UPnP-Device Bridge, Service Face, PDA UI Plug-in …), version information, etc.

- Various tool supporting testing services (e.g. Virtual Devices, UPnP Tools, prepared test UPnP scripts …)
- Preconfigured Siemens Smart Home platforms, e.g. Service Gateway Platform and Service Enabling Platform. It allows a developer to set a local Siemens Smart Home system and thus allows for the testing of implemented services

Service Development Download Manager (SDDM)
The SDDM module serves as a mediator between the service developer environment and software components stored in the SDR. Via the SDDM a programmer can download all required data for service development. The other functions of the SDDM are:

- developer authorization and authentication
- automatic update mechanism (once the developer has downloaded all the services from the SDR, he needs only to synchronize at times his database with the SDR)
- handle notification from the NSUM component about new or updated services

New Service Upload Manager (NSUM)
After development of new a service or change of existing service, a developer can use the NSUM API to upload new service files to the ASDS system.

Service Development Download Client (SDDC)
The SDDC module resides on the service developer server and is connected to the one or more SDDM modules at the operator's side. Basically it downloads all required software components and stores it in Local Service Development Repository (L-SDR). The SDDC module is responsible for:

- Downloading of service components from the various SDRs components (SDDC can be connected to more than one SDDM)
- Performing automatic updates upon change or addition of new services to the SDR
- Automatic downloading of necessary components (when an implemented service uses other services)
- Downloading of components, stored in the SDR, for testing purposes, e.g.: service implementation files, virtual device implementations, service proxy implementation, platform configuration files, etc.

Service Development Upload Client (SDUC)
Once a developer has implemented a new service, a SDUC module will upload all the necessary software parts to the operator's database.

Local Service Development Repository (L-SDR)
The Local SDR is used to store all software components downloaded form the operator's server, thus they do not need to be downloaded each time a new service is developed.

Eclipse and Visual Studio Integration Manager Plug-ins (EIM & VSIM)
The Integration Manager plug-ins extend the functionality of common Integrated Development Environments, such as Eclipse or Visual Studio. For this purpose, it makes use of data stored in Local Service Development Repository. Some example features of the Integration manager include:

- Pop-up and help windows while using service classes and APIs,
- Service browser – to search for various services APIs and get their specification
- Automatic inclusion of needed components, when (jar files, dll, etc.)
- Automatic inclusion of dependency information (on other service components, versions …)
- Auto completion code support
- Additional IDE views and windows for Siemens Smart Home Service development
- Support for development of specific Siemens Smart Home UI

6.4.2 Many operator and service developers

The idea of the SDSS system is to enable developers to use already implemented services. When an existing service has been developed by an other company or operator, the SDSS allows the connecting to this company and downloading of all necessary components. However, when there are many operators and developers hosting their own service databases, searching for any service and downloading components might be a problem.

The solution of this problem could be a centralised system created by all interested parties (e.g. Smart Home Consortium). The system would keep the data about services and other software components developed by interested parties. After the authentication and authorisation procedure a developer can search for any service he is interested in. When he decides to download some software parts the system will automatically redirect a connection to the operator's or developer's server (where the real data are stored).

6.4.3 Other issues

While implementing the SDSS system one should have in mind several issues.

Legal rights
Normally the source code of a service should not be available for other companies. In the case of binary components some companies can also restrict their availability. They might not want their software to be used for developing other services for free

Integration with SEP
The Service Enabling Platform on the operator's side already has a lot of functionality which might be used by the SDSS system (e.g. authentication, authorisation, download and upload modules, databases, etc…)

Sensitive services
For some services, like security services, it might be better for safety reasons to restrict their API availability. In this case only trusted developers and applications would have access to this API. Some solutions can be adopted from the mobile industry, where an unsigned mobile application has no access to some platform API.

Dependency
Some standards can be used for software components to describe and check dependencies (e.g. versions, configuration …)

Future interaction
Actions of different services can have contradictory effects. For example, an energy-saving service will turn off the lights when nobody is at home and a security service simulating the presence of inhabitants will turn on different lights. Therefore, a system for confects management should be considered (e.g. using various service priorities).

6.5 Service Architecture Construction tool

As already stated, a programmer should start the development with a decision about the type and architecture for a service. In this activity a *Service Architecture Chooser* can be helpful. However, it can happen that the architecture proposed by a tool does not match exactly the developer's needs. The Service Architecture Construction tool aims to address this problem and allows a programmer to redefine service architecture in a convenient way.

Figure 6.11 presents what an example Service Architecture Construction tool could look like. On the picture all hardware parts existing in the Siemens Smart Home are drawn. These parts are for operation as well as for testing purposes. After a developer has chosen service architecture a proper set of software components is marked (checkboxes near each component). On the figure the marked set corresponds to the service type – Manual Device Control. With this GUI a programmer can simply change the set of components he wants to implement for the Siemens Smart Home service. This idea can be further extended by implementing a drag and drop mechanism. In this case a designing will be similar to designing a GUI in popular IDEs. A set of icons symbolizing particular Siemens Smart Home hardware and software elements will be available to the developer. He will then drag and drop needed components (like in standard GUI design tools) and thus will build service architecture.

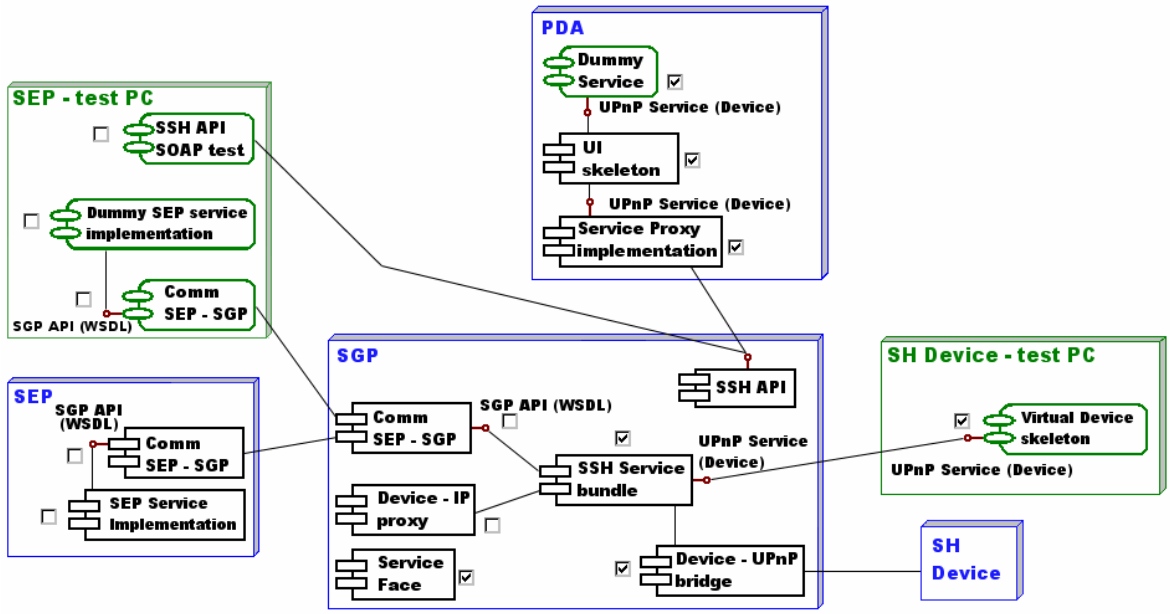


Figure 6.11: Service Architecture Construction tool

The picture shows what an example Service Architecture Construction tool could look like.

7 Example tool implementation

In this chapter an example tool (Siemens Smart Home Service wizard) for Siemens Smart Home service development is presented. The tool is implemented as an Eclipse plug-in. The program is described and a justification for chosen solutions is given.

7.1 Eclipse plug-in

> *Eclipse is a free software/open source platform independent software framework for delivering what the project calls "rich-client applications", as opposed to "thin client" browser – based applications. [www.wikipedia.org]*

The platform is based on the OSGi standard and defines a plug-in architecture, which means that Eclipse can be very easy extended by some new features. In fact, all the functionality exposed by the platform to a developer is achieved by plug-ins. Usually Eclipse is delivered with many standard plug-ins, which create advanced IDE functionality (e.g. automation tools, debugger, GUI editor).

Since the platform is free and it is easy to extend its functionality, there are already many plug-ins available on the market (free and commercial ones). Furthermore, the Eclipse Foundation strongly supports the platform and the Eclipse community. This results in many tutorials, services, complementary products and support available in the Internet.

For the reasons mentioned above, Eclipse is widely used among developers in many companies. Also during the development of the Siemens Smart Home prototype at Siemens Corporate Technology, Eclipse was the main development IDE for java code. Therefore, I have decided to use this platform for implementing one of a tool proposed in previous chapter.

7.2 Siemens Smart Home Service wizard

When a programmer starts a new project in Eclipse he has several wizards to choose from. The most generic one is a simple java project wizard, which creates some standard eclipse project files. For a development of smart home services, where there are many software components and possible configuration files, it would be helpful to have an additional wizard. The wizard can create all Siemens Smart Home service specific files. Therefore, it would be a good starting point for the inexperienced as well as for experienced Siemens Smart Home service programmers. Therefore, I have decided to implement the Siemens Smart Home Service wizard, which can encapsulate many other useful tools presented in the previous chapter (see figure 7.1).

After a developer chooses the Siemens Smart Home Service Wizard, he is asked to enter basic project data such as service name or a package. In the next window a service type must be chosen (figure 7.2). If a programmer does not know what service type to select, he can press a button and use a Service Type Chooser Tool (see chapter 5.2.6).

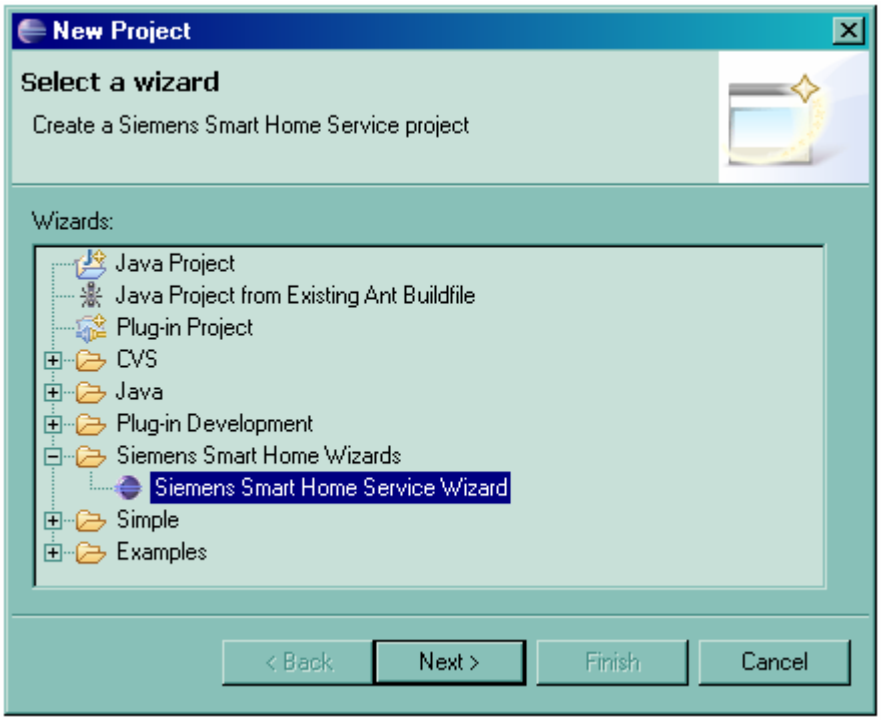


Figure 7.1: Siemens Smart Home Service wizard in Eclipse

The screenshot presents how a developer can start a new project using the Siemens Smart Home Service wizard.

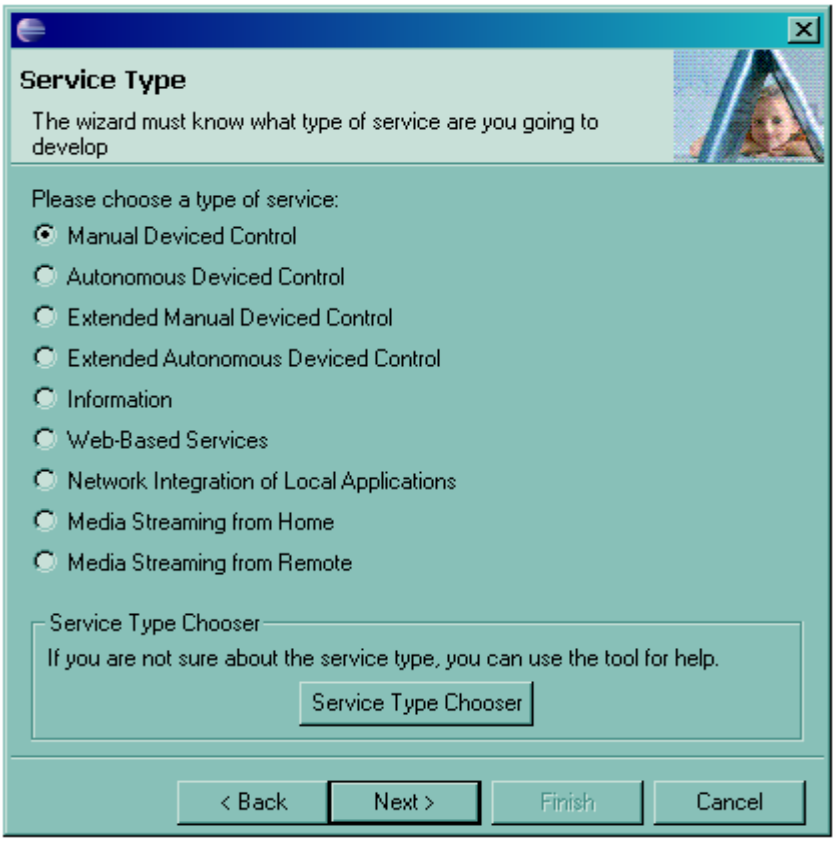


Figure 7.2: Siemens Smart Home Service wizard – Choosing a service type

The screenshot presents various service types to choose from. If a developer does not know what kind of service is he going to implement he can use the Service Type Chooser tool.

The next screen presents the architecture for the implemented service (figure 7.3). For the prototype version of this wizard the presented picture of the architecture is static and for all service types the same. As already pointed out in chapter 6.5, it can be further improved. The developer sees all possible hardware and software components existing in Siemens Smart Home architecture. According to the selected service type, a subset of those components is selected for implementation (marked checkboxes). At this place it is possible to change the service architecture by checking / un-checking checkboxes.

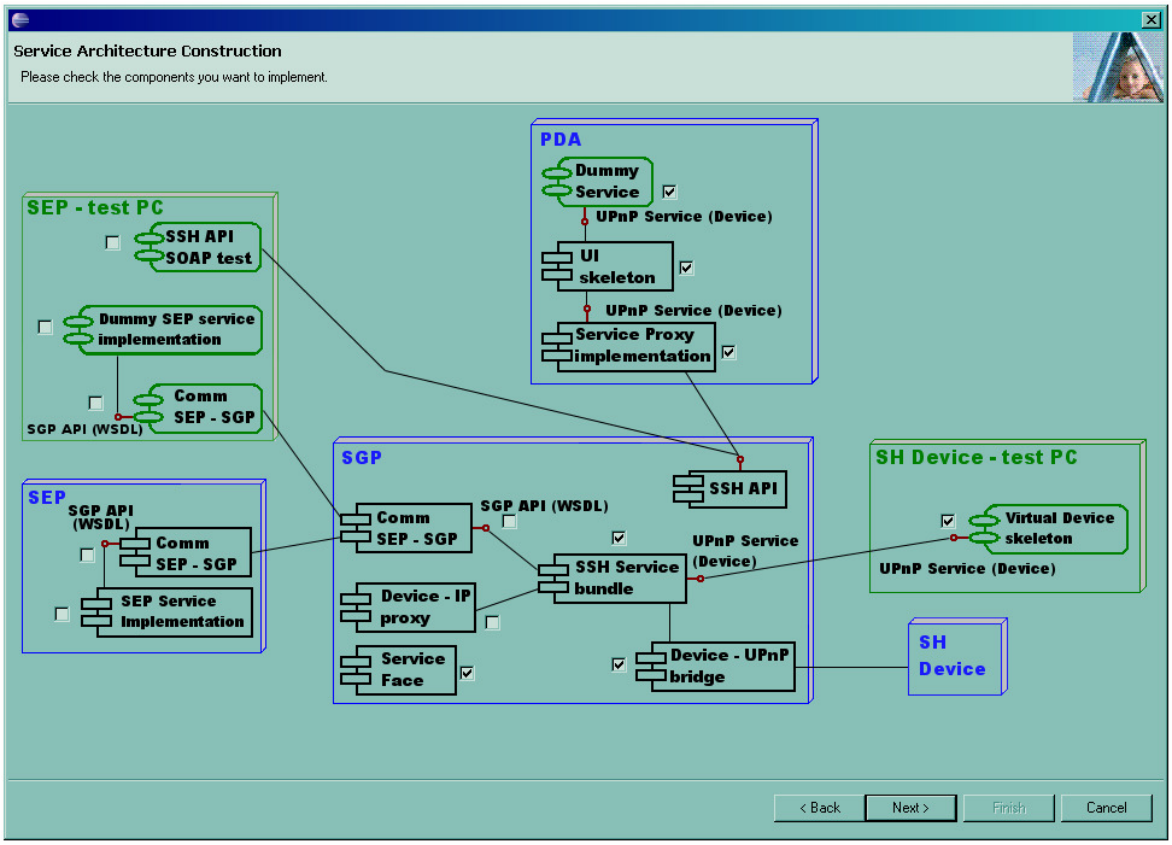


Figure 7.3: Siemens Smart Home Service wizard – Service Architecture

The screenshot presents Siemens Smart Home architecture. Various software components (or skeletons) are automatically generated when a corresponding check-box is selected.

In the next wizard window, the last one, some additional information of the service should be given. Depending on the type of implemented service a path to the UPnP description file must be provided (not when implementing Web-Service or Information service type). Additionally, if a part of a service will run on the backend a WSDL file should also be provided. This information will serve as an input to generate various service software components.

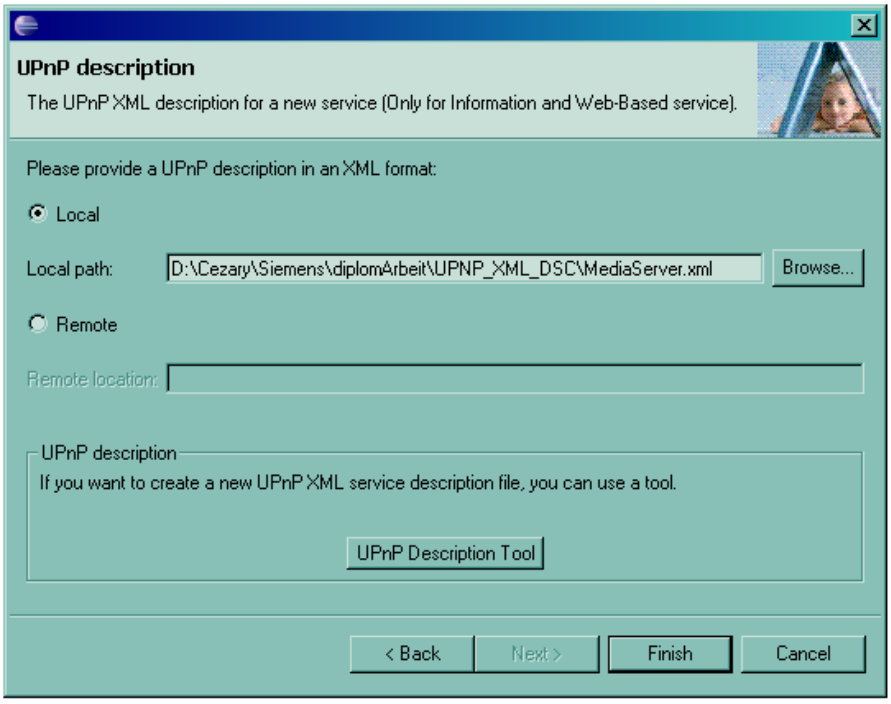


Figure 7.4: Siemens Smart Home Service wizard – Service Architecture
The screenshot presents the last wizard page where a user is asked for the UPnP XML description file.

7.2 XSLT based approach

The result of the Service Wizard is a new Siemens Smart Home Service project in the Eclipse workspace. The project should contain some Siemens Smart Home service configuration files, a structure of packages and directories for implementing service parts, and source code of software components (or its skeletons) where it was possible to create them automatically. The most important and sophisticated part is the automatic generation of those components. For this purpose the eXtensible Stylesheet Language Transformations 2.0 (XSLT) technology was used [22]. XSLT was chosen due to a number of advantages such as easiness to transform XML files, widely accepted standard in industry, ability to change XSLT sylesheets without recompiling the code.

Firstly, the input data for the generation are in form of Extensible Markup Language (XML), i.e. UPnP description file and Web-Service description in WSDL. Since the XSLT was specially designed for transforming XML documents into other formats, XML stylesheets can be easily processed.

Secondly, the XSLT stylesheets are not compiled binaries like Java or C++ code. The XSLT processor takes both data in XML format and the XSLT file and performs transformations (see figure 7.5). This nature of interpreted languages allows changing code without recompiling the whole program. For the service development it means that the XSLT stylesheets can be easily changed matching new requirements for the design of service components.

Another advantage of the XSLT standard is its wide acceptance and use in industry. This results in many XSLT tools and engines which make the whole development process with XSLT much faster and easier. For example, the project SAXON [53] provides a free XSLT 2.0 processor with simple API for java. The SAXON processor was integrated into the Service Wizard to generate a number of service software components.

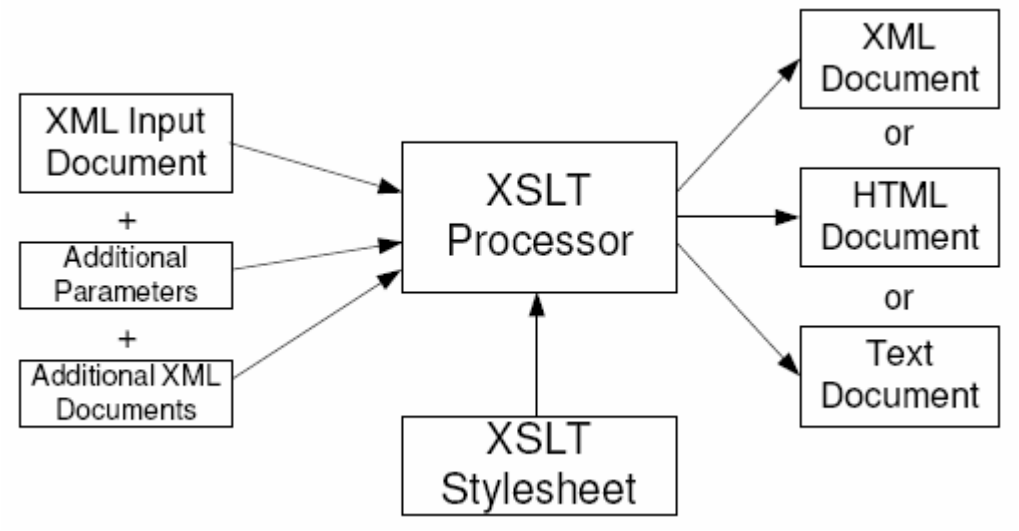


Figure 7.5: XSLT processor

The diagram depicts the XSLT transformation process. The XSLT processor takes XML document and XSLT stylesheet as an input and can produce various output documents.
Source: Hamburg-Harburg University of Technology; Inter-Object Communication lecture [57]

7.3 Virtual Devices generator

I have chosen to implement the generator for Virtual Devices after speaking with a service developer team. The meeting was organized in order to gather requirements from the service developers, gain some hints and suggestions about possible tools. As a result, they propose the ranking list with a number of tools which might be needed at most. The ability to test some service logic without having real security devices has one of the highest proprieties.

The Virtual Devices generator makes use of the UPnP device description file in XML format. It than generates a device java file which implements all the services specified in the UPnP device description. For each UPnP service a corresponding java file is generated. It includes all methods and arguments defined in the UPnP description. After that a developer should fill the skeleton files with logic corresponding to the device behaviour.

The wizard creates a new project in the Eclipse workspace directory and produces source files for a number of the Siemens Smart Home service components. These service components were previously chosen by a developer during the service architecture design phase. Currently, only the Virtual Devices generator is implemented, so under the sources directory are only java classes corresponding to this component (see figure 7.6). Additionally, all UPnP XML description files are copied to the UPnPServiceDescripton directory and a ".project" file containing basic data about the Siemens Smart Home project (e.g. service name) is created.

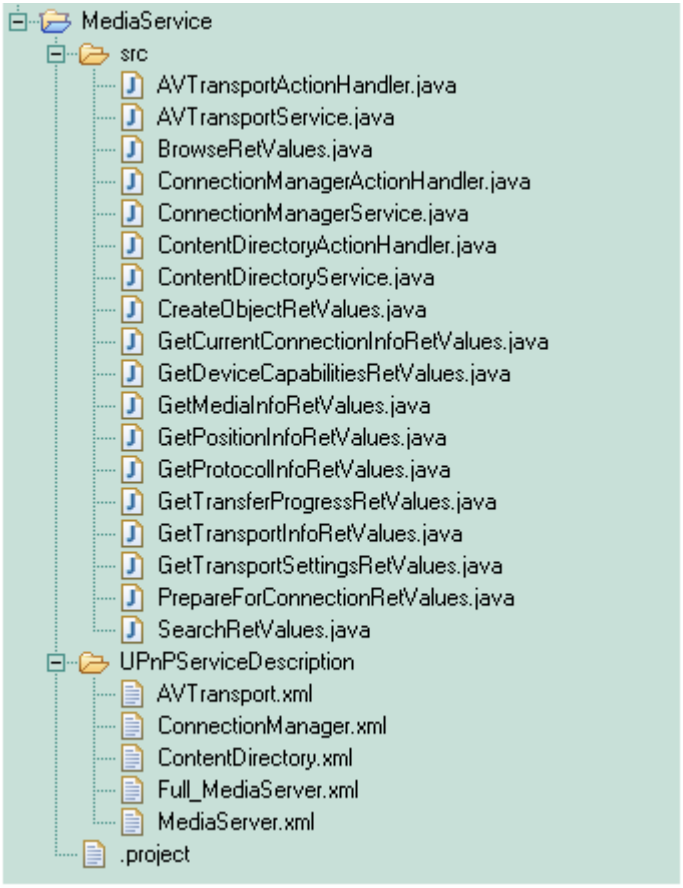


Figure 7.6: Siemens Smart Home Service Project in Eclipse

The picture presents part of the Eclipse workspace with a Siemens Smart Home project. For its purpose various directories and files were generated.
Example: UPnP Media Server

On the figure 7.7 examples of source code is presented. At first a UPnP XML device description file has to be available. The file is transformed, via the XSLT processor, according to the ruled specified in XSLT stylesheet transformation file. In the example below a Java service interface source code for AV Transport service is created.

Figure 7.7: Code examples

The picture presents an example of source code. Using a XSLT transformation stylesheet a java service interface is generated (based on a given UPnP service description).

7.4 Plug-in architecture

Here Siemens Smart Home Service wizard architecture is presented.

7.4.1 Eclipse platform

Eclipse platform architecture is designed to enable the easy creation and integration of plug-ins, which contribute to Eclipse by adding some special feature. The platform is built in layers of plug-ins, where each plug-in connects itself to extension points defined by the lower-level and thus provides some additional functionality. At the same time these plug-ins expose theirs extension points for higher-level layers.

The basic part of the platform is the runtime core. It implements the runtime engine that starts the platform base and dynamically discovers and runs plug-ins. All other functionality is supported by a number of basic plug-ins delivered together with Eclipse:

- Resource management
- Workbench UI
- Team support
- Debug support

- Help System
- Java Development Tools (JDT)
- Plug-in Development Environment

From the point of view of the Siemens Smart Home Service plug-in, the most interesting is the Workbench UI part. It implements the workbench UI and defines a number of extension points that allow higher level plug-ins to contribute menu and toolbar actions, drag and drop operations, dialogs, wizards, custom views and editors.

The Siemens Smart Home Service plug-in contributes to a workbench wizard extension point. This makes the implementation easier since all the actions that launch the wizard are already set up by the workbench. The only think one has to do is to supply the wizard that will be used. A standard wizard is composed of a three main components (see picture 7.8):

- Wizard Dialog – top level dialog in wizard; defines the standard wizard buttons and manages a set of pages that are provided to it; the dialog and the button management activity (e.g. enabling and disabling of the Next, Back and Finish buttons) is provided by the platform.
- Wizard – controls the overall appearance and behaviour of the wizard, such as title bar text or image; a common practice is to extend a Wizard class, which provides some typical behaviour; a programmer's task is to add wizard pages and implement the behaviour that should occur when the user presses the Finish button.
- Wizard Page – defines the controls that are used to show the content of the wizard page; it responds to events in its content areas and determines when the page is completed; the WizardPage class provides an implementation of basic behaviour of a wizard page; a developer has to extend the class in order to provide some GUI controls and determine whether a user has supplied enough information to complete the page.

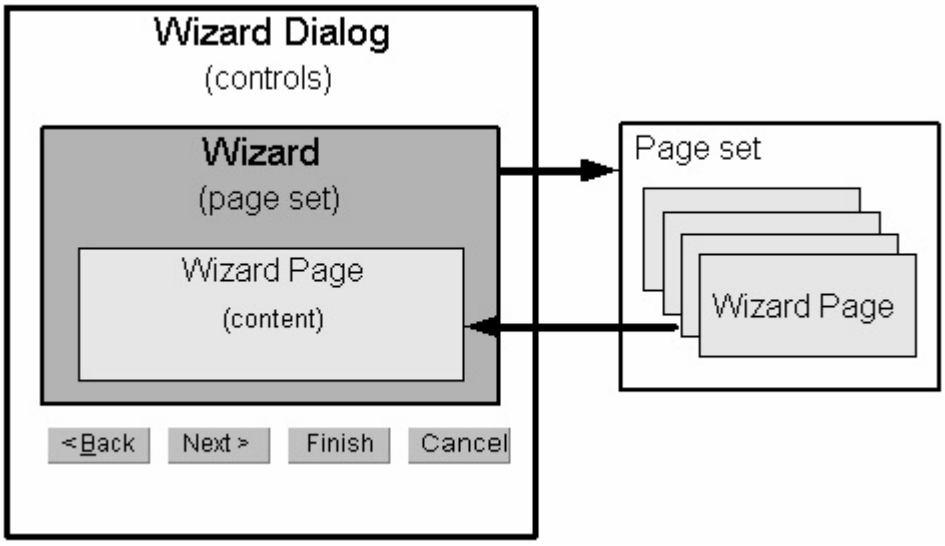


Figure 7.8: Wizard components in Eclipse

The picture presents wizard architecture in Eclipse. It consists of the three main elements: the Wizard Dialog, the Wizard and the wizard Page.
Source: Eclipse tutorial [10].

7.4.2 Siemens Smart Home Service wizard architecture

For the purpose of the Siemens Smart Home Service wizard the Standard Widget Toolkit (SWT) and JFace UI framework were used extensively. The SWT is widget toolkit for java developer that provides a portable API (like a Swing toolkit). It was developed especially for Eclipse and is not as powerful as Swing. However, due to its lightweight architecture it is faster and less complex to use. The JFace UI framework provides a higher-level application construct for supporting dialogs, wizards, actions, user preferences and view management.
The Siemens Smart Home Service wizard source code is organized in to three packages:

- com.siemens.ct.sshservice
- com.siemens.ct.sshservice.wizard
- com.siemens.ct.sshservice.wizard.generators

com.siemens.ct.sshservice
The package includes the SshServicePlugin class which extends AbstractUIPlugin (a must for every plug-in). It provides implementation for basic OSGi bundle operations, e.g. start, and stop.

com.siemens.ct.sshservice.wizard
This package contains all the classes which are responsible for interaction with a user, e.g. wizard class itself and wizard pages.

com.siemens.ct.sshservice.wizard.generators
The package contains all classes involved in the generation process. The generation is performed when the user presses the Finish button on the Siemens Smart Home Service wizard dialog.

Wizard architecture
The class diagram of the wizard is presented on the figure 7.9. The main class, SshServiceWizard, hosts all wizard pages and manages the flow between them (Back and Next buttons). When the user provides all necessary information, the Finish button is enabled and a programmer can create a new service project. Methods performFinish and doFinish take care of the creation process. For source code generation the wizard uses classes from generators package.

Currently there are four wizard pages:

- SshServiceMainWizardPage – for providing a service name and package
- SshServiceTypeWizardPage – asks the user about the service type he wants to develop
- SshServiceComponentsWizardPage – presents all the components existing in the Siemens Smart Home architecture; a programmer has the ability to select/deselect service parts he wants to implement; at the beginning some components are selected based on the service type; for building a SshServiceComponentsWizardPage the Jigloo Eclipse plug-in was used [19]. Jigloo is free for non-commercial use SWT/Swing GUI builder for Eclipse
- SshServiceUPnPDscWizardPage – for providing a UPnP description file

For keeping the service project data a ProjectCreateData class is used. Various wizard pages store in this class the data provided by a programmer. At the end this information is used by the main wizard class to create a new service project.

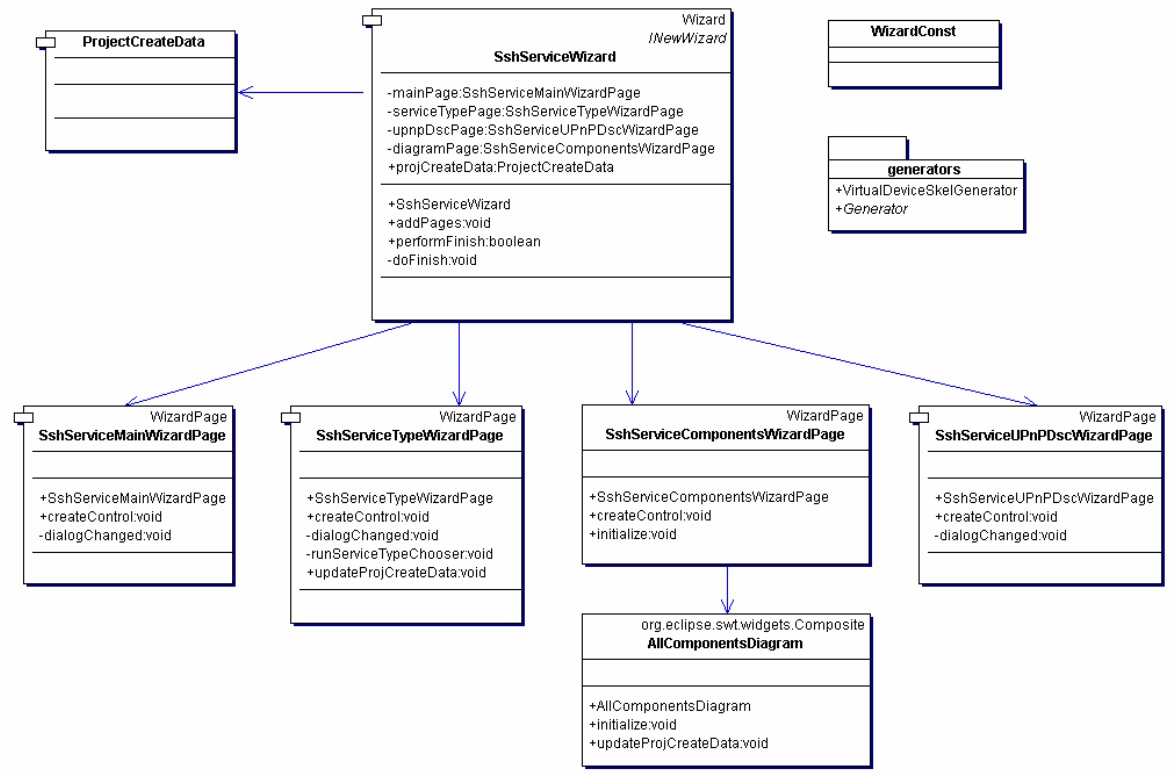


Figure 7.9: Siemens Smart Home Service wizard class diagram

The diagram presents the class diagram of the Siemens Smart Home Service wizard.

Generators

For each service component a developer wants to implement there is a corresponding generator class. This class handles the automatic creation of the source code. For example, in figure 7.10 a VirtualDeviceSkelGenerator is responsible for generating a skeleton code for Virtual Devices. The class extends the abstract Generator class and is responsible for actual XSLT transformations. The VirtualDeviceSkelGenerator is responsible mainly for supporting proper parameters (e.g. output file name, source file, XSLT file name). To extend the current wizard a programmer has to provide other generator classes.

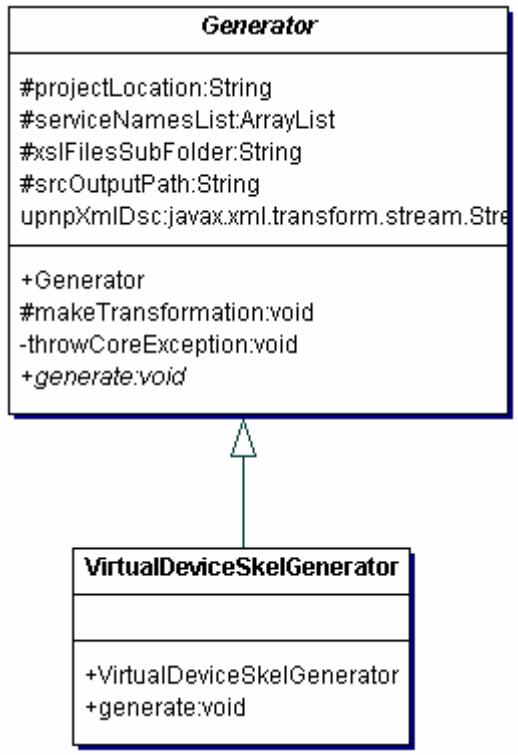


Figure 7.10: Generators package

The picture depicts two classes which are contained by the package com.siemens.ct.sshservice.wizard.generators.

8 Case study – service development

In this chapter the development process of two services; Device Control and Scene Control is presented. The development method, used tools, and various testing are based on research performed in three previous chapters. This case study assumes the use of the Siemens Smart Home Service wizard tool for Eclipse platform and service type chooser tool developed by J. Wechsler [64].

8.1 Device Control service

This service enables the remote control of individual devices and groups of devices from various interaction devices. In the current development stage the supported control devices are PDA (rich client platform) and PC (web – thin client platform). The controlled devices that have to be added into the Siemens Smart Home are European Installation Bus devices [12], which include:

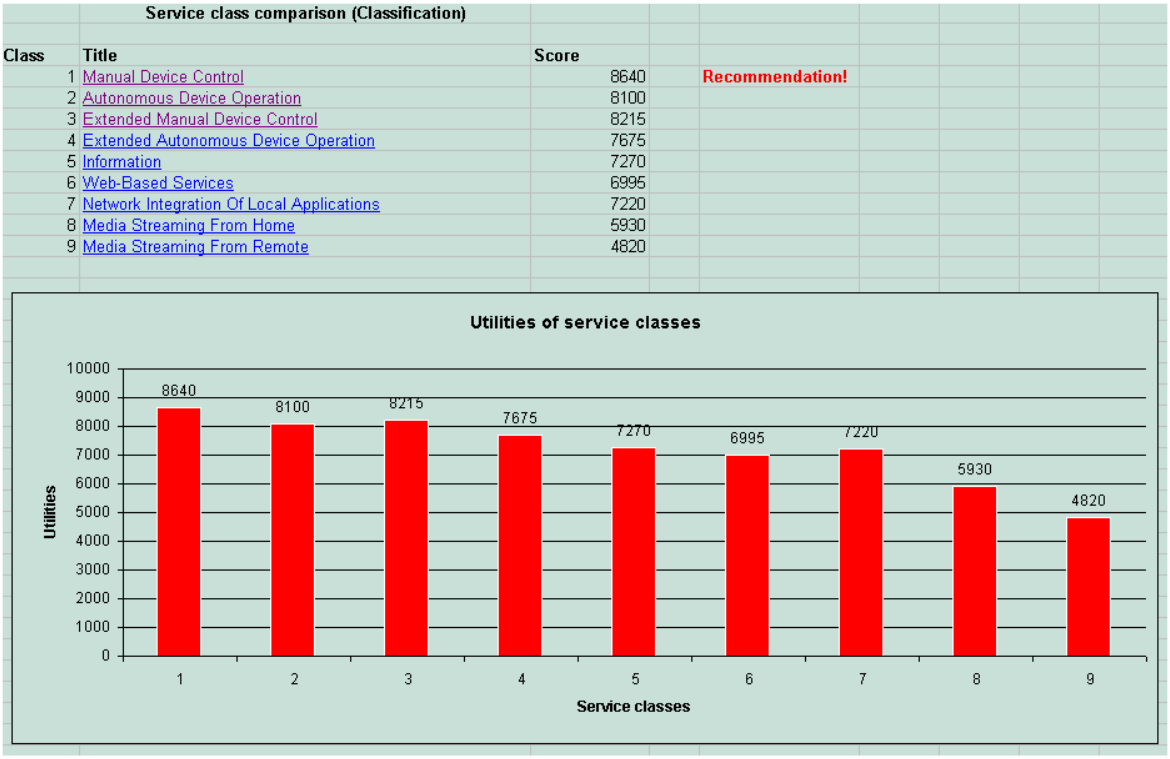
- load switchers (for switching on and off a binary light),
- dimmers,
- roller blinds controller,
- temperature controller.

Inhabitants should be able to control those devices (e.g. set a new dimming value of a specified light) via a GUI. They should also be able to check the current status of a device (e.g. check if the light in the kitchen is on), as well as become events upon the state change.

8.1.1 Architecture

In order to develop a new service, a programmer starts a new project in Eclipse using the Siemens Smart Home Service wizard. At the beginning, on the first wizard page, the developer is asked for the service name (e.g. DeviceControl). Next he has to choose which service type is going to be developed. At this stage it is possible to use J.Wehsler tool [64], which is especially helpful if a programmer is new in smart home domain.

After firing the architecture selection tool the developer has to answer a set of eighteen questions such as interaction with actuators and/or sensors, use by other services, frequency of user interaction at home, etc. For simplicity the answers are made in the form of a combo box, so the developer needs to choose a proper from the set. Having all the questions answered the tool makes a ranking of possible service types (see figure 8.1). Thus the top service types are: Manual Device Control (8640 points), Extended Manual Device Control (8215) and Autonomous Device Operation (8100). The highest score is granted to Manual Device Control and it suits the Device Control service well. The developer has to choose the radio button corresponding to suggested service type and can proceed further.



Service class comparison (Classification)

Class	Title	Score	
1	Manual Device Control	8640	Recommendation!
2	Autonomous Device Operation	8100	
3	Extended Manual Device Control	8215	
4	Extended Autonomous Device Operation	7675	
5	Information	7270	
6	Web-Based Services	6995	
7	Network Integration Of Local Applications	7220	
8	Media Streaming From Home	5930	
9	Media Streaming From Remote	4820	

Figure 8.1: Service Architecture Chooser tool

The screenshot is from the Service Architecture Chooser tool. After answering a set of questions, possible service architecture are presented to a developer. Here a Device Control service is going to be developed.

The next wizard page shows the general Siemens Smart Home service architecture (please refer to chapter 7.2). All the software components which are relevant to previously chosen service type are checked for implementation. It means that for these parts the entire code or a skeleton will be automatically generated. For example the Service bundle (on the SGP) and Service Proxy Implementation component (on the PDA) are chosen. The tool should generate a skeleton for the first one and the entire code for the second one. The other, not selected software parts, will be omitted in service implementation process as not relevant in this case. At that stage the developer can still change service architecture by selecting / unselecting the service components (checkboxes on the wizard page).

The selection made by the wizard meets the requirements of the Device Control service. These software components are (see figure 8.2):

- Siemens Smart Home Service bundle (SGP)
- Device –UPnP bridge (SGP)
- Service Face (SGP)
- UI skeleton (PDA)
- Service Proxy implementation (PDA)
- Dummy Service (PDA – for testing)
- Virtual Device skeleton (test PC)

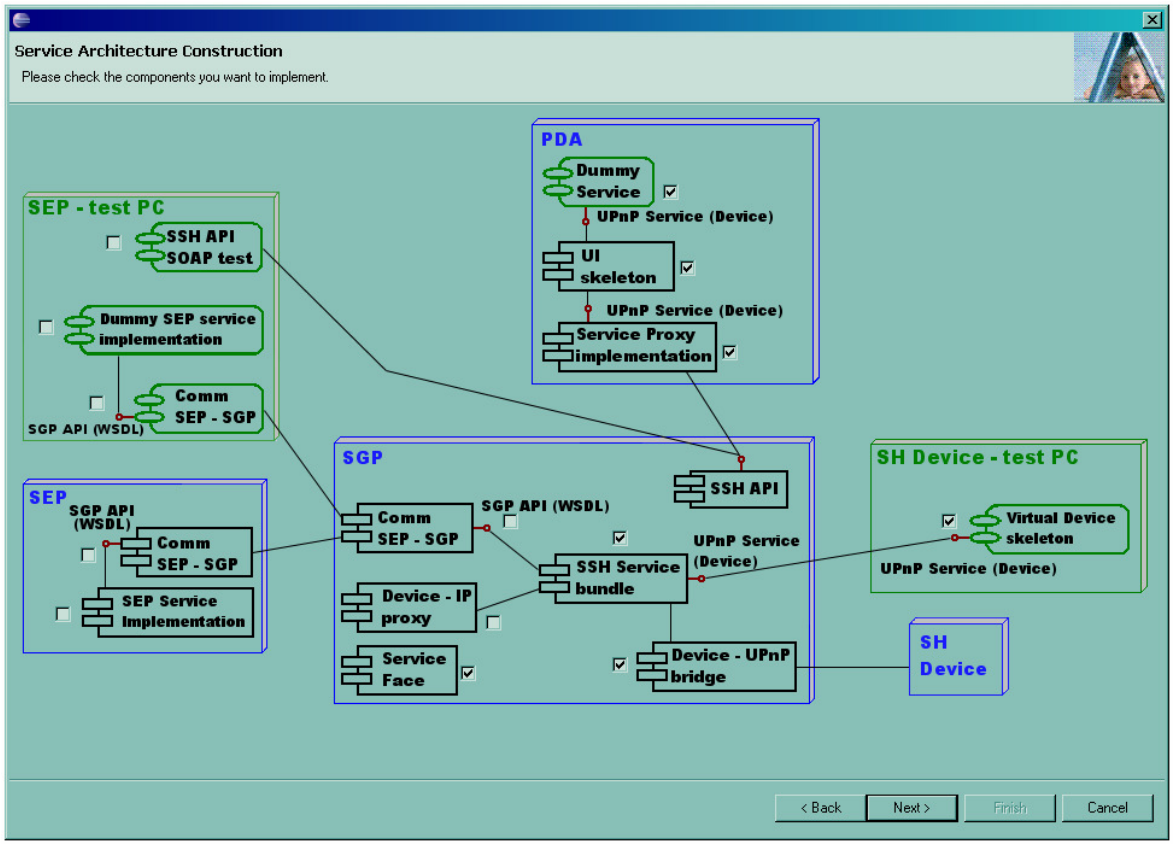


Figure 8.2: Siemens Smart Home Service wizard – Service Architecture

The screenshot presents the Siemens Smart Home Service wizard page where the developer can select / deselect service components. The proposed selection was made by the wizard and corresponds to the Device Control service type.

8.1.2 Development process

The general service development process discussed in chapter 6 can now be adjusted to Device Control service, as shown on figure 8.3. The first step, i.e. service architecture selection, has already been made. The next step is to provide an UPnP service/device description in XML format. Based on this document all the necessary components will be generated. This includes: Device UPnP Bridge skeleton, UI Service Face skeleton, Service Implementation skeleton, UI skeleton (C#), and Service Proxy implementation. Additionally for testing purposes, Virtual Device skeleton will be created.

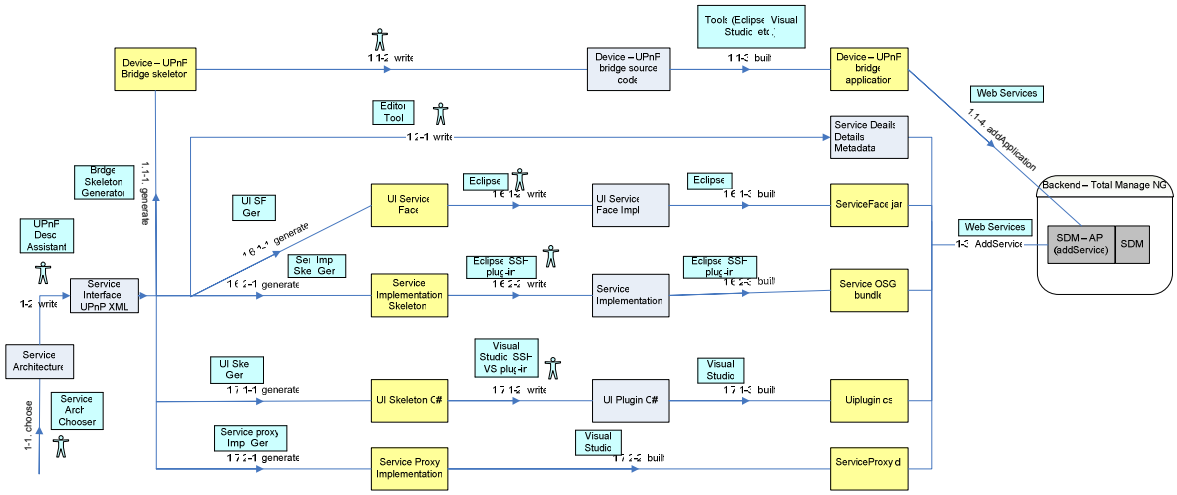


Figure 8.3: Device Control service development process

The diagram presents Device Control service development process. Only the relevant software components are shown.

8.1.3 UPnP description

When the developer makes the final decision about service architecture he can proceed further. In the next and the last wizard page an UPnP description of the Device Control service has to be provided. The description serves as an input document for various generators. The UPnP organization (UPnP) has already standardized many common services and devices. These documents are available on the web page and are free to download. Therefore, the programmer should check first the availability of the corresponding description documents. In the case of Device Control service the devices (binary light, dimmable light, roller blinds and temperature controller) are already standardized. The document has to be adjusted to the specific devices, i.e. EIB devices in this case. In the UPnP standard of a device some actions are mandatory, some are optional and some might be vendor dependent. The adjustment aims at selecting optional actions, which are supported by real devices and eventually adding action which correspond to additional functionality of devices (vendor dependent actions).

Having provided a valid UPnP service/device description document, source code of developing service components is generated.

8.1.4 EIB - UPnP Bridge

The WIB devices do not implement UPnP protocol. Therefore, there have to be proxy software EIB-UPnP-Bridge, whose main task is to make EIB devices reachable by UPnP protocol. The Bridge is responsible for storing and updating states of the objects associated with the EIB devices. The bridge receives UPnP messages and events, translates them to the EIB messages and sends them to the EIB bus. Similarly, the application receives messages and events form the EIB bus, translates them to the UPnP messages and generates UPnP events. The Bridge runs on the computer, which is connected to EIB bus (e.g. SGP). The connection to the EIB is done via Ethernet.

To simplify the development of the EIB - UPnP Bridge a skeleton can be generated. It will cover most of the UPnP part of the application. The developer has to fill the code with application logic and program the EIB part, receiving, sending and handling EIB messages and events from the bus.

Additionally, for testing purposes a Virtual Devices application skeleton has to be generated. The software imitates real devices on the test computer, and thus it enables performing unit testing of a service running on the gateway without real devices.

8.1.5 Siemens Smart Home Service bundle

In the case of the Device Control service, application logic is programmed inside the EIB-UPnP Bridge. The only thing one should do is create java stub classes and interfaces for new devices and services. These classes have to be then integrated into Siemens Smart Home API, which means that corresponding ".class" files have to be included in package com.siemens.upnp.generated.device. The purpose of the stubs is to provide the interface for other service components and bundles. If other services will use the functionality exposed by UPnP devices they can refer to it using generated classes. The stubs can be generated automatically by the Device Builder delivered with the Siemens Smart Home platform.

Another issue is to program the EIB-UPnP Bridge as a service bundle instead of a stand alone application. This will include, as described above, the generation of a skeleton code. Additionally, the skeleton must include some OSGi bundle specific code.

According to the Siemens Smart Home specification and User Interface Extensibility Platform (UIEP), the web UI is realized via a concept of UI Service Face (see chapter 6.2.1). A UI Service Face skeleton will be automatically generated and filled with logic by a programmer.

8.1.6 PDA - UI Plug-in

For PDA, where the rich UIEP is deployed, three components have to be implemented: UI plug-in, Service Proxy and Dummy Service Proxy. The first one provides a UI interface of a new Device Control service. The source code in C# can be partly created by a generator. It needs then to be imported to Microsoft Visual Studio and completed with logic and GUI design. For GUI, a programmer can use already developed widgets and colour schemes. The Service Proxy component is a mediator between the UI plug-in and the SGP and is used by the plug-in to call service methods. It is be fully generated by a tool. The last component Dummy Service Proxy is also automatically generated. It enables performing GUI tests without the need for communication with the SGP.

8.1.7 Testing

In this chapter the testing activity for Device Control service components are discussed. First, each single software components (figure 8.4) are checked during unit tests (e.g. EIB devices). In the next step various parts, like EIB-UPnP Bridge and OSGi service bundle, are checked together (middle level on the diagram). In the end, all components are tested in terms of their proper operation and seamless interoperation (top level).

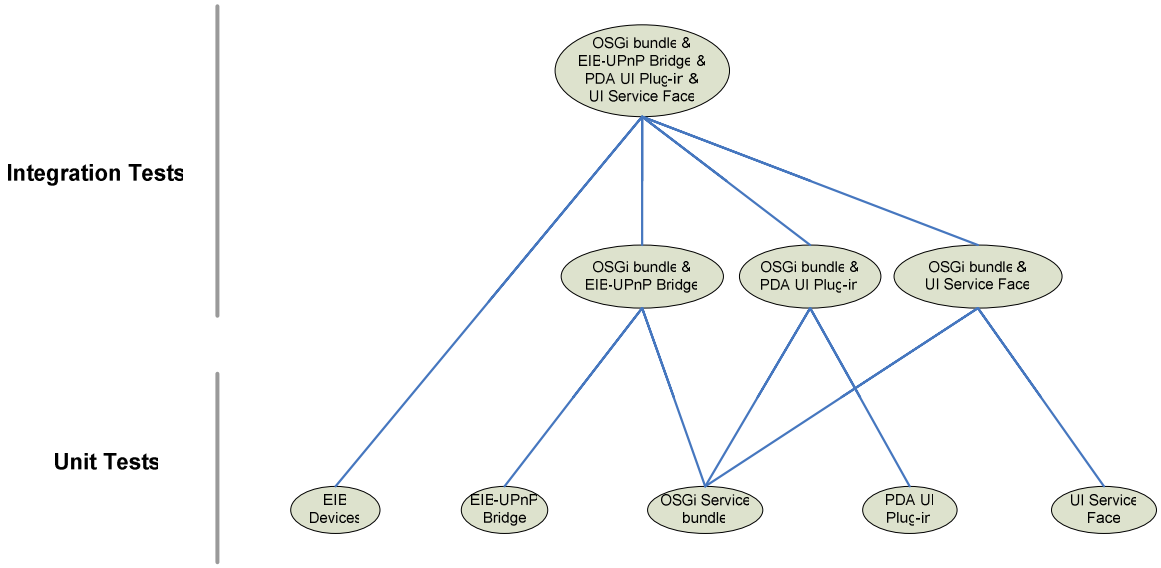


Figure 8.4: Testing Device Control components

The diagram depicts test activities for Device Control components. At the lower level unit tests are performed. The middle and higher level presents integration tests.

Unit tests

At first, all of the developed components should be tested in isolation. Below is the description of unit tests which has to be performed during the development of Device Control service.

EIB Devices

The EIB devices can be tested with ETS3 software (see chapter 5.2.3). The software provides a monitoring tool where one can send messages to devices (turn on the light) and receive notification when, for example, somebody switches off the light using a wall switch. In this way a proper configuration and operation of all devices can be checked.

EIB-UPnP Bridge

The proper operation of the Bridge can be checked with an UPnP general control point (see chapter 5.2.1). The tool discovers all the UPnP devices in a network and enables to fire actions and receive events for the UPnP devices. The Bridge can be tested by firing UPnP action on the control point and observing whether the expected effect on a real EIB device took place. The same can be done for events, by manipulating manually an EIB device and checking when the proper notification comes to the control point.

PDA UI Plug-in

The first test can be made using a PDA simulator provided by Microsoft Visual Studio software and Dummy Service Proxy. In this way a proper application operation and lack of logical errors can be checked. The next step is to test the UI plug-in on the real PDA. In this phase there is still no connection to the SGP and the Dummy Service Proxy is used to simulate real service actions.

OSGi Service bundle

In order to test OSGi Service bundle in isolation Virtual Devices and Smart Home service test tool has to be used. Using the tool (see chapter 6.3.1.3), various Web Services action of the implemented service on the SGP can be invoked and checked for correctness.

UI Service Face
The UI Service Face can be tested on a PC and other interaction devices from the point of view of GUI. The presented web interface should look on each interaction device corresponding to the developer intention.

Integration tests

After unit tests the service components must be tested together. This includes checking the proper operation of two components. At the end all of the software parts should cooperate seamlessly.

UI Service Face & OSGi service bundle
To check those two components a developer has to use Virtual Devices. Using a test PC, or other interaction device which supports web UI, Device Control service actions are invoked and checked for correctness.

PDA UI plug-in & OSGi service bundle
This interaction test is similar to the previous one. The difference is that instead of UI Service Face (web UI) a UI plug-in on the PDA is tested (rich client UI).

OSGi service bundle & EIB-UPnP Bridge
In this case the cooperation between the OSGi service bundle and the EIB-UPnP Bridge is checked. The Virtual Devices from the previous tests are substituted for the Bridge and real devices. To test new service a SOAP/Web Services tool is used. It allows firing service API via Web Services and thus checking the proper functionality.

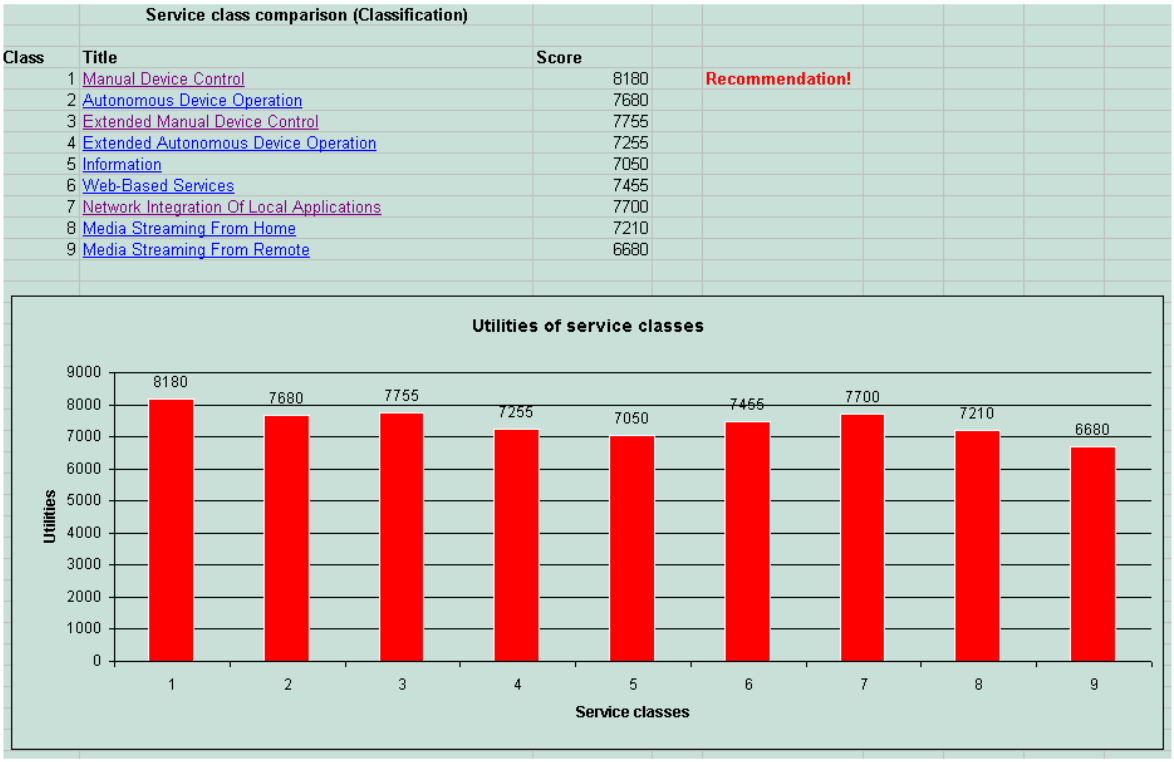
OSGi service bundle & EIB-UPnP Bridge & PDA & Web UI
In this step all developed service components are tested together, i.e. OSGi service bundle on the SGP, EIB-UPnP Bridge together with real EIB devices, UI plug-in on the real PDA, UI Service Face with all web UI interaction devices.

8.2 Scene Control service

A scene is a sequence of commands that are executed together. For example a living home scene allows a user to switch off all lights and defined home appliances as well as to arm his security system by pushing one button before leaving home. A Scene Control service should allow a user to define and program various scenes, such as: leaving home, entering home, dinner, romantic, chill out. The programming of a scene includes defining a set of commands, e.g. for a romantic scene it might be: switching off all the lights, dimming one lamp to 20%, closing roller blinds and playing suitable music. The user should be able to fire previously programmed scenes via a GUI from various interaction devices.

8.2.1 Architecture

As in the previous example (Device Control service) the Siemens Smart Home Service wizard is used to develop the Scene Control service. When a developer has to decide which type of service he is going to develop he can use J.Wehsler's tool [64]. After answering the same set of question as in the first example, the tool ranks all of the possible service types (see figure 8.5). This time the top service types are: Manual Device Control (8180), Extended Manual Device Control (7755) and Network Integration of Local Applications. The tool has recommended Manual Device Control service type and this suits well the Scene Control service.



Service class comparison (Classification)

Class	Title	Score	
1	Manual Device Control	8180	Recommendation!
2	Autonomous Device Operation	7680	
3	Extended Manual Device Control	7755	
4	Extended Autonomous Device Operation	7255	
5	Information	7050	
6	Web-Based Services	7455	
7	Network Integration Of Local Applications	7700	
8	Media Streaming From Home	7210	
9	Media Streaming From Remote	6680	

Figure 8.5: Service Architecture Chooser tool

The screenshot is from a screenshot of the Service Architecture Chooser tool. After answering a set of question possible service architecture are presented to a developer. Here a Scene Control service is going to be developed.

Since the service type is the same as in the Device Control service, also all components the developer needs to implement are the same (see figure 8.2). However, unlike in the previous example, the architecture presented by the next wizard page does not suit the Scene Control service. The service does not introduce any new home devices, and thus the UPnP-Device Bridge and the Virtual Devices should be omitted. These components can be simply deselected using wizard GUI. The proposed service class is still the best choices\ for Scene Control service (among the classes defined by J.Wechsler). At the end the programmer has to implement following software parts:

- Siemens Smart Home Service bundle (SGP)
- Service Face (SGP)
- UI skeleton (PDA)
- Service Proxy implementation (PDA)
- Dummy Service (PDA – for testing)

8.2.2 Development process

The development process of Scene Control service is similar to the one discussed for Device Control service (figure 8.3). However, since there are no new home devices to be integrated, a Device-UPnP Bridge component is not needed (most upper process path on the diagram). Similarly Virtual Devices do not need to be developed. For testing purposes they should be already implement while introducing other services (e.g. Device Control service, Security service …).

8.2.3 UPnP description

The next step is to provide an UPnP Scene Control service description in XML format. After looking at the UPnP web site it is clear that the service has not been standardized yet. This means that

the developer has to write the description from scratch. The document in XML format has to follow certain syntax defined by UPnP organization. For UPnP service description a standard procedure is to define service actions. For Scene Control the actions might be: InvokeScene, CreateNewScene, DeleteScene, ChangeScene, etc… Each function takes input arguments (0 or more) and returns output arguments (0 or more). The parameters should also be defined in the description documents (e.g. argument type, allowed values, default value). For the creation of the service description one can use a Device Scriptor tool from Intel (described in chapter 5.2.1).

The UPnP organization with time is standardizing more and more devices and services. It is very probable that the Scene Control service will be also standardised in the future. When it happens the service should be adopted to the new UPnP description.

8.2.4 Siemens Smart Home Service bundle

For Scene Control service the application logic is concentrated in the Siemens Smart Home Service bundle. Based on the UPnP description the tool can create a skeleton which includes the standard code for the SGP, OSGi bundle and UPnP service. A programmer has to provide the application logic.

Similar to the previous service a UI Service Face skeleton is generated and the developer has to provide a Scene Control specific code.

8.2.5 PDA – UI Plug-in

This step is also the same as in Device Control service. For the rich UIEP platform on a PDA three components have to be implemented: UI plug-in, Service Proxy and Dummy Service Proxy. For the first one a source code is partially generated and needs to be then completed with a GUI design. Service Proxy and Dummy Service Proxy are automatically created by a tool.

8.2.6 Testing

The testing processes for Scene Control and Device Control service are to a large extent alike. The difference is that for the Scene Control there are no tests of the UPnP-Device Bridge or real home devices (e.g. EIB devices). Another aspect is the provision of Virtual Devices for unit and integration tests. The service assumes the use of other services, e.g. Device Control, Security, and Entertainment. After a user fires a scene, corresponding actions should be executed on real devices. For example Living Home should result in switching off all lights at home and arming the security system. Therefore, for unit and integration tests the Virtual Device applications are needed. The programmes already written when developing “core device services” can be used. In this place a Service Development Support System, described in chapter 6.4, is very helpful. It assumes the existence of a repository with all service components, which might be used for the development of new services.

9 Evaluation

This thesis works out a method for development Siemens Smart Home services. As a part of it, various domain specific tools are introduced. These tools were specially developed to simplify the service development process. For the practical part of this thesis, a simple Eclipse plug-in was developed. The plug-in implements some example tools and enables to create a new Siemens Smart Home service project in the Eclipse workspace.

In this chapter the evaluation of the proposed tools as well the Eclipse plug-in is presented. For this purpose a number of people, closely working with the development of the Siemens Smart Home, were ask for feedback. At the current stage of the Siemens Smart Home project, it was not yet possible to obtain feedback from third-party developers of a security service.

9.1 Introduction

After the start of working on this thesis, a meeting with developers was organized. After presenting the developer team the tool chain, I gathered feedback from them. Generally, they liked the idea of most of the tools. Particularly they found a Virtual Device and Virtual Device Skeleton Generator the most interesting. At that time the team was trying to implement a security service and they have problems with getting all needed devices. The Virtual Device tool would help them a lot, since it would be possible to test the service without having real security devices installed.

The developer team also proposed a new component *Device IP Proxy* on located on the Service Gateway Platform. The component should enable the direct access to devices at home, which are normally behind the NAT. For the Security Service it would apply to a special camera at home.

9.1 Evaluation preparation

In order to evaluate the development process and the Eclipse plug-in, people developing Siemens Smart Home Platform were delivered a packet of data. In consists various documents such as:

- Eclipse plug-in Installation instruction
- Development tools and process description
- Evaluation instruction
- User questionnaire (see Appendix)
- Eclipse plug-in .jar file
- Description files for the Scene Control service
- Description files for the Media Server service

The users were asked to make a new Siemens Smart Home service project using the implemented wizard. As an input for generator they had a choice of two different service description files: the Scene Control service and the Media Server service.

9.2 Feedback

In this chapter feedback gathered form the users is presented.

9.2.1 Tools

The users were asked to rate each tool in terms of usefulness. The result was deferent with the ones obtained form the service developer at the beginning. Most of the components and tool especially designed for testing purpose were given low grades (as opposed to the feedback gathered at the beginning of this work). These components are:

- Virtual Devices and Virtual Device Skeleton Generator
- Dummy Service Proxy Implementation and Generator
- Dummy SEP Service Implementation and Generator

Moreover the Device Proxy Generator tool, proposed by service developers of the security service at the beginning, was found by most of the users as not useful.

The best rates got the applications which are strictly connected with the most important components is Siemens Smart Home architecture. These components are present almost in every type of service. The tools very useful are:

- Editor Tools
- Service Implementation Skeleton Generator
- UI Skeleton Generator (for PDA)
- Service Proxy Implementation Generator (for PDA)
- Communication SEP-SGP (and SGP-SEP) Generator

They evaluated the Service Development Support System as a very interesting and valuable system. However, they remarked that the system is very complex and it might be difficult to implement it. Since there are a lot of entities involved in Siemens Smart Home architecture (many 3rd party developers, smart home operators and Siemens as a platform deliver company) a proper place to host the SDSS main system is still an open issue.

9.2.2 Implementation

The general expression about the implemented prototype is positive. However a number of remarks and suggestions were given.

The UI could be improved and polished up in order to make it more easy to use. First, the page which lists all the service types should contain more information about the services. For example one service type explanation sentence can be added close to a radio button. Also the way the Service Architecture chooser tool is called form the wizard is not convenient. The next page shows Siemens Smart Home architecture and a check box close to each component. At the first glance it is not clear which service components are selected. This could be improved by blurring the unselected service parts.

Moreover, to support service development for inexperienced programmers a concept of cheat sheets can be implemented. The cheat sheets are a kind of an interactive help. A user, who wants to do a particular task, is instructed step by step. When he makes a progress the help information is adopted to a new situation (showing new options and possibilities).

After a user presses a *Finish* button a new project is generated, which should be informed with a short information message. The project structure and the files in the Eclipse workspace have one drawback. There should be more comments about generated classes, methods, variables, etc. This information would improve the readability for developers new in the Siemens Smart Home domain. One user suggested also that the OSGi bundle activator class name should be short and meaningful, e.g. *Activator*. The longer name, *MediaServerActivator* is less clear and can not be found at first

glance. Moreover, there lack of classpath to JVM and to OSGi libraries makes the project unable to compile.

For better understanding of the generated components a short *readme.txt* file can be provided. It should list all the created files, their short description, and the hints about missing parts (which a programmer must implement).

9.2.3 General

In general all users found the proposed tool chain useful for a Siemens Smart Home service development. They concluded that the main benefit of the tools is faster service development. Additional advantage is the ability to implement a service for users inexperienced in Siemens Smart Home architecture.

To make the programming of a service easier for new programmers a short PPT presentation about Siemens Smart Home architecture, service types and wizard would be appreciated.

At least the Eclipse plug-in implements two tool: Virtual Devices Skeleton Generator and Service bundle Implementation Skeleton Generator. To fully benefit from the designed tool chain other applications have to be provided.

10. Conclusion

The service development method, worked out in this thesis, gives a programmer clear directions for implementing a new Siemens Smart Home service. This incorporates a number of supporting software tools. Some of these applications are already available on the market, while others were specially designed for the purposed of the development process.

This chapter summarizes the achieved results and discusses their contribution to Siemens Smart Home as well as to other domains. Moreover, the limitations of the thesis are described as well as possible improvements.

10.1 Contribution

In this section the usefulness of the analysis presented in this master thesis are evaluated. The importance of the studies is also judged for distributed, heterogeneous environments other than Siemens Smart Home.

Service development process overview
This master thesis provides analysis on service development activity for Siemens Smart Home environment. Based on the service type, different service architectures are proposed and different sets of components must be implemented. Those components are often developed in different technologies and deployed on various hardware devices (heterogeneous, distributed environment). At first, this activity is presented using UML use case diagrams and is compared with general software development process (chapter 4). In the next chapter the block diagram presents the exact steps a programmer has to go through to develop a smart home service.

General development tools
Many software development applications available on the market are described. These tools support programming, testing and maintenance of software components and can be used for development of software programs in any domain. In the case of the Siemens Smart Home the tools are placed in the service development process diagram. It gives a reader an overview where in the process a particular tool is used.

Siemens Smart Home domain specific tools
The next part described tools which are specific for the Siemens Smart Home. Those applications are closely related to architecture, particular technologies and design concepts of the Siemens Smart Home. At first, already existing applications are introduced (e.g. UPnP tools, Service Architecture Chooser). In the next step service development process is analyzed and additional tools are proposed. This includes many simple source code generators and testing programs. Usually a skeleton (or entire source code in case of simple components) is generated from the UPnP device description data or WSDL document. Firstly, it saves a developer time on writing a repeatable code. Secondly he does not have to study Siemens Smart Home architecture in detail in order to implement a service. The programmer needs only to write code in certain sections of generated skeletons.
Furthermore, a concept of Service Development Support System (SDSS) is presented. It is a system which supports the implementation and testing of a new service through a number of features (e.g. extends IDE functionality, support implementation of services which are dependent on other services, classification of services …).

Tool prototype
A simple Eclipse plug-in is implemented as a realization of some proposed tool concepts. The Siemens Smart Home Service wizard tool supports creating new services. After providing some data

about the new service, a wizard will generate an Eclipse project with source code for implemented service components. Currently only generation of Virtual Device skeleton is supported.

Relevance for other domains
The process of service development is similar to some extent for different distributed, heterogeneous environments. In these environments there are a number of hardware platforms with different software systems installed. The service is distributed over those platforms, which means that various parts of the service are implemented in different technologies (like in the Siemens Smart Home). Chapter 2 introduces some R&D projects in the smart home domain as well as in other domains. Some of the technology used in these projects are the same as in the Siemens Smart Home (e.g. OSGi, .NET CF).
Therefore, basic concepts of service development described in this master thesis (e.g. unit testing of components) and general support tools (described in chapter 5) can be used in other domains. However, service classification, detailed service development process for the Siemens Smart Home, generated source code and test components have to be adopted for a new environment.
The more sophisticated Service Development Support System (SDSS) was described from the high perspective. The main architecture concepts and functional futures were presented in terms of Service Oriented Architectures. Therefore, the tool design is not directly connected with the Siemens Smart Home and can be implemented for other domains.

10.2 Limitations

This section presents some limitations concerning the service development method worked out in this thesis.

Evaluation by third-party service developers
Currently, no large-scale service development is in progress for Siemens Smart Home. Thus, a full evaluation is currently not possible.

Better service type classification method
Cased studies presented in Chapter 8 showed that the method for choosing the service type is not perfect (the architecture for Scene Control service needed some corrections). Therefore the service types and automatically generated service architectures should be better adapted to the Siemens Smart Home domain.

Development process presented from high perspective
The development process described in this work is presented from the high perspective. Based on the service type a developer is instructed what components he has to develop. However, there are no detailed guidelines how to develop particular components (e.g. OSGi service bundle). In this case the developer needs reference to Siemens Smart Home specifications which describe specific Siemens Smart Home architecture parts in more detail. For the OSGi bundle development a programmer can refer to existing documentation and tutorials.

10.3 Future Work

In this section possible improvements and extension of this work are presented.

Many tools not implemented
The proposed tool chain includes many new applications useful in service development activity. Since only a small number of them were implemented, other tools should be written in the next step. For new generators this process is easy and mainly means writing new XSLT stylesheets.

Iterative evaluation
The service development method and tool chain have to be evaluated by real developers during implementation of a new Siemens Smart Home service. Moreover, the evaluation process needs to be performed iteratively over a longer period of time. Only in this way is it possible to develop a useful, correct and convenient tool chain for service development

Improvement of the process (UI Service Faces and SEP Service)
While writing this thesis the development of the Siemens Smart Home solution was still in progress. Therefore, there was no clear description how UI Service Faces and service parts running on the SEP have to be implemented. Future work should include more detailed analysis on these components.

Tool adaptation for other domains
In order to use the chain tool in other domains most of the applications need to be redesigned. For the implemented Eclipse plug-in the service types and their classification method have to be redefined for a new environment. In the next step the corresponding service architecture must be redesigned, since the new domain architecture and the service components might differ. Next the generators must be rewritten. When XSLT approach for source code generation is continued, this consists mainly of writing new XSLT stylsheets.

A Appendix

A.1 User questionnaire

As part of the proof of concept, several users were asked to verify the usefulness of this work for Siemens Smart Home service development. A set of question presented below served as a basis for the evaluation.

1. Approach
1.1 What do you think abut the idea of using a tool chain for service development?

2. Development process
2.1 Is the development process correctly designed/documented (diagram, descriptions)?
2.2 Do you find it useful as a guideline for the service development?

3. Tools (please find description of tools and components in Tools_description.doc file)
3.1 Do you think that the new tools are helpful?
3.2 Can you rate the tool in terms of usefulness (1 – very useful, 5 - not useful)?

Implementation tools \ Score	**1**	**2**	**3**	**4**	**5**
Service Architecture Chooser					
UPnP Description Assistant					
Bridge Skeleton Generator					
Editor Tools					
UI Service Face Generator					
Service Implementation Skeleton Generator					
Communication SGP-SEP (and SEP-SGP) Generator					
Device Proxy Generator					
UI Skeleton Generator (for PDA)					
Service Proxy Implementation Generator (for PDA)					
SEP Service Skeleton Generator					

Testing tools \ Score	**1**	**2**	**3**	**4**	**5**
Virtual Devices & Virtual Device Skeleton Generator					
Dummy Service Proxy Implementation & Generator					
Dummy SEP Service Implementation & Generator					

3.2 Do you know any tools available on the market which can be useful for service development (and which were not listed in this work)?
3.3 Do you have any ideas about tools which can be useful for service development (and which were not listed in this work).
3.4 Do you thing that the Service Development Support System (SDSS) might be useful?

4. Implementation
4.1 What do you think about implementing tools as Eclipse plug-ins? Is it good idea? Should it be developed for other development platforms?
4.2 Is the GUI of the Siemens Smart Home wizard satisfactory? Do you find it easy to use?
4.3 Have you any ideas what improvements should be made to UI?
4.4 Is there anything in the plug-in that you did not like (was not clear, not convenient to use)?

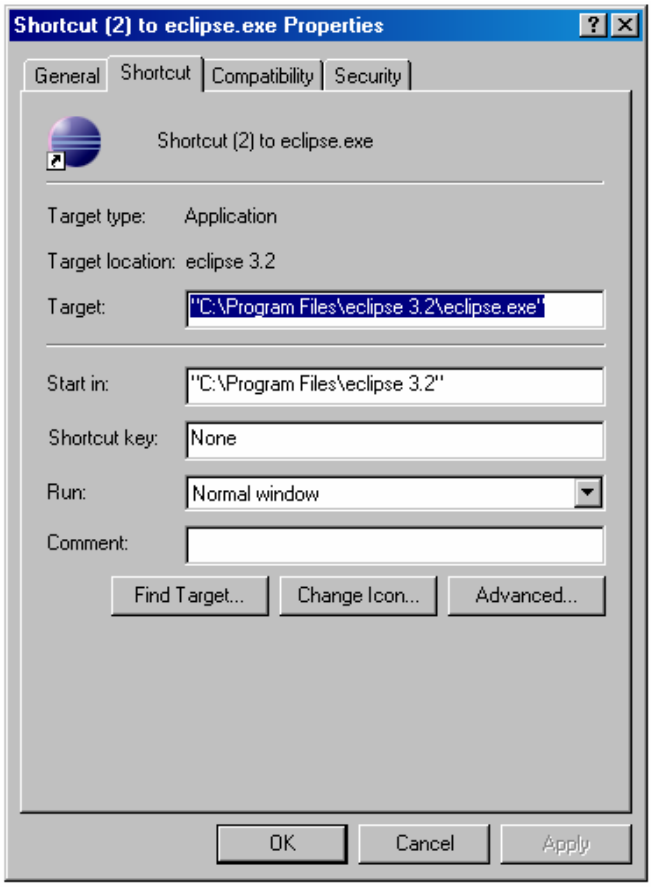
5. Generated project
5.1 Does the created Eclipse project structure meets a developer needs?
5.1 How do you find the structure, form and content of generated files?

6.1 General
6.1 What are the main benefits of this chain tool?
6.2 Is the tool helpful for somebody who does not know Siemens Smart Home architecture in details?
6.3 What do personally find useful in terms of service development

A.2 Installation Instruction

The Siemens Smart Home service wizard (Eclipse plug-in) was implemented as a practical part of this thesis. The steps necessary to install the plug-in are presented below.

1. The plug-in works stable with Eclipse 3.2
2. Unpack the SshServiceWizardPlugin_distribution.zip package. It contains following components:
    a. ***installation_instruction.doc*** – this document
    b. ***evalution_instruction.doc***
    c. ***Tools_description.doc*** - the description of various tool and the diagram of Siemens Smart Home Service development process.
    d. ***User questionnaire***
    e. **plugins** – directory with eclipse plugin
    f. **saxonb8-7-3j** – jar files or XSLT processor SAXON
    g. **MediaServer** – UpnP description files for Media Server service
    h. **SceneControl** - UpnP description files for Scene Control service
    i. **ServiceTypeChooser** – Service Type Chooser application (an Excel file)
3. Copy file *com.siemens.ct.sshservice_1.0.0.jar* from *plugins* directory into */eclipse/plugins* directory.
4. Modify eclipse.exe shortcut on your desktop.
   Right click on the shortcut and choose properties. Go to shortcut tab and modify the *Target* property.
   Currently it can look like similar to this:
   "C:\Program Files\eclipse 3.2\eclipse.exe"



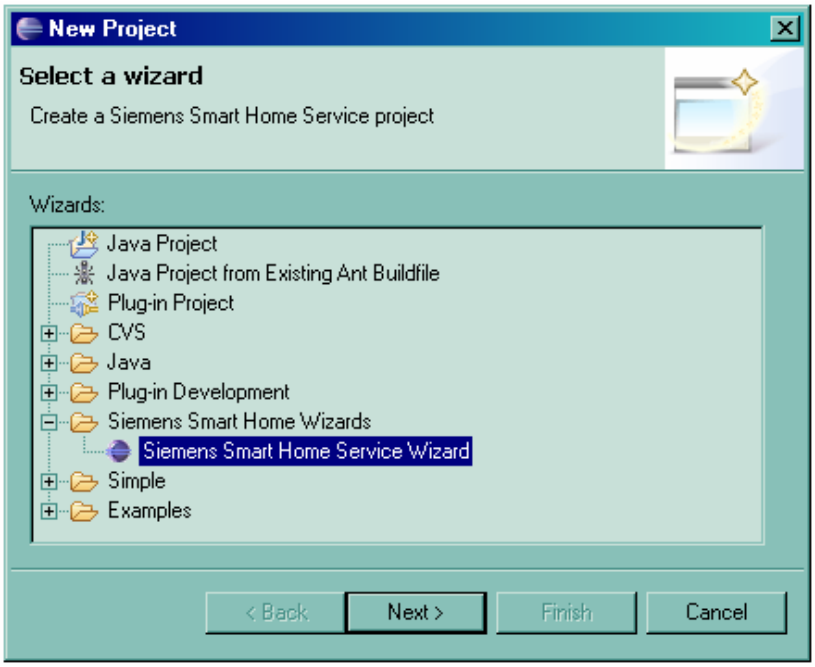
Change it to:
"C:\Program Files\eclipse\eclipse.exe" -vmargs -Djava.endorsed.dirs=D:\ SshServiceWizardPlugin_distribution \XSLT_SAXON\java\saxonb8-7-3j

The path should point to the folder where the SAXON jar files are (see point one). On my computer it was D:\ SshServiceWizardPlugin_distribution \XSLT_SAXON\java\saxonb8-7-3j.

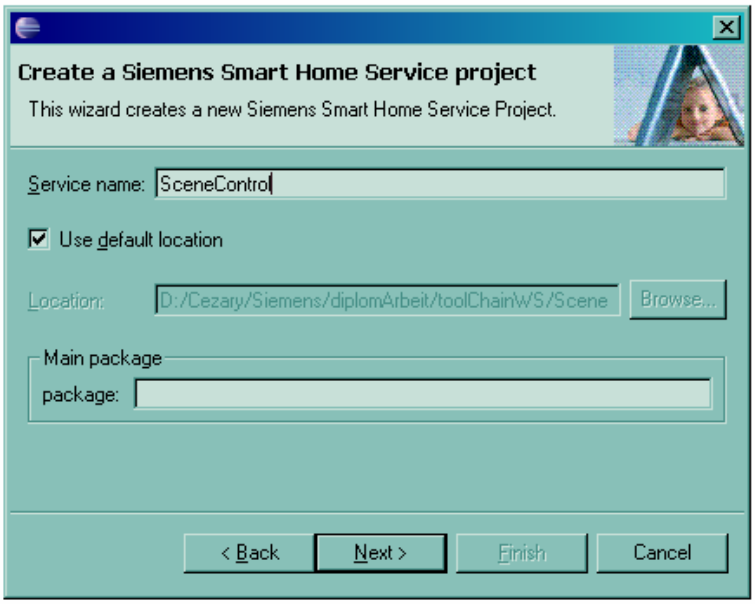
A.3 Evaluation Instruction

Several Siemens Smart Home developers were asked to evaluate the Eclipse plug-in. In this section a short plug-in tutorial and the evaluation instructions are presented.

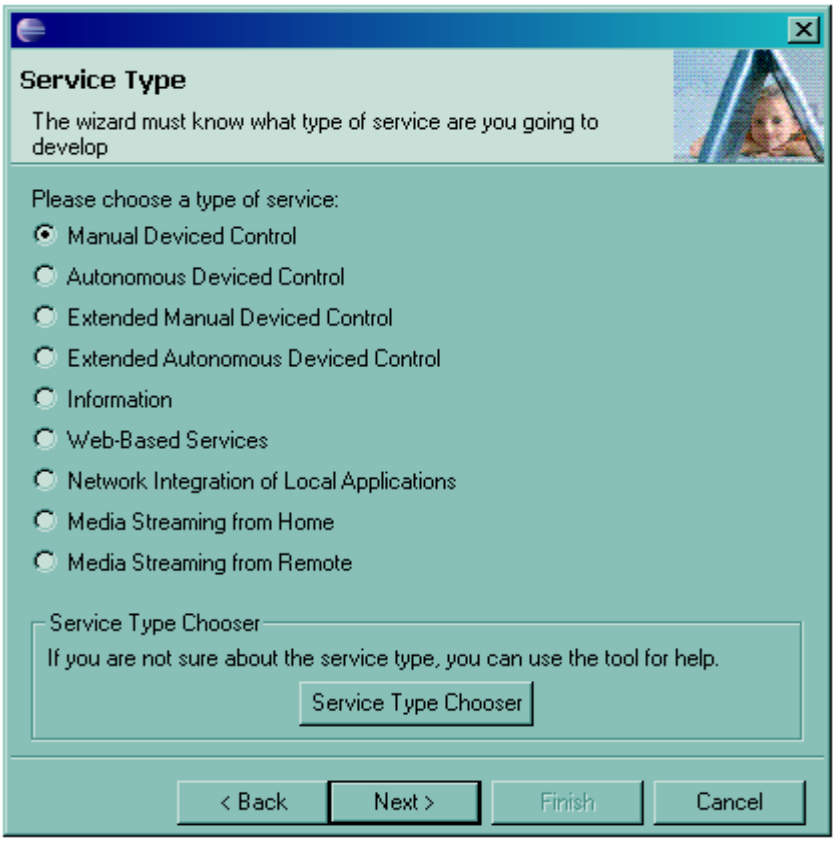
1. After having installed the plug-in run Eclipse platform.
2. Run Siemens Smart Home Service wizard.
   Go to *File->New->Project*. Choose Siemens Smart Home Wizards - > Siemens Smart Home Service Wizard.



3. In the first page of the plugin provide service name.



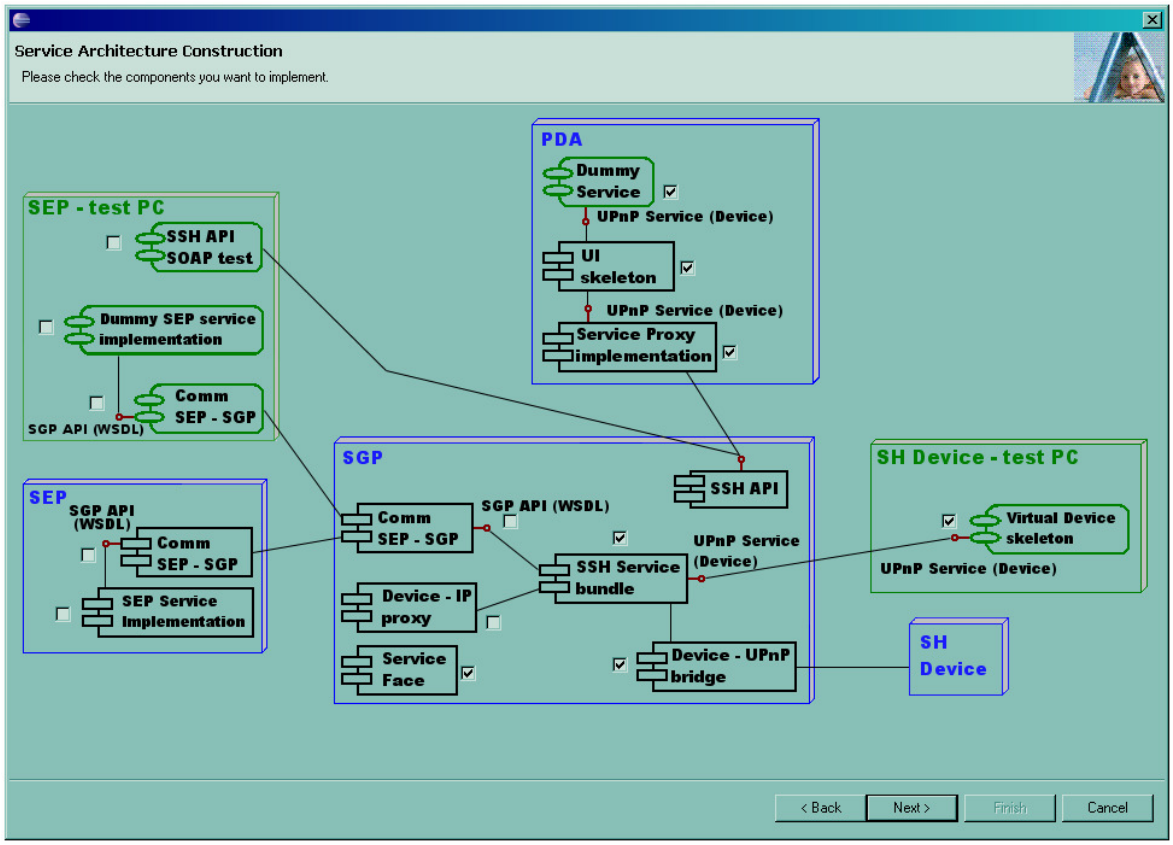
4. In the next page choose the service type you want to implement. Use *Service Type Chooser* tool if you are not sure about the type. If you use the tool you will be asked for the path to Excel and to the tool (the tool is located in the …SshServiceWizardPlugin_distribution\ServiceTypeChooser\ArchitectureSelectionMethod.xls)



5. On the next page you may decide which service components to implement. The default setting is selected (for the previously chosen service type).

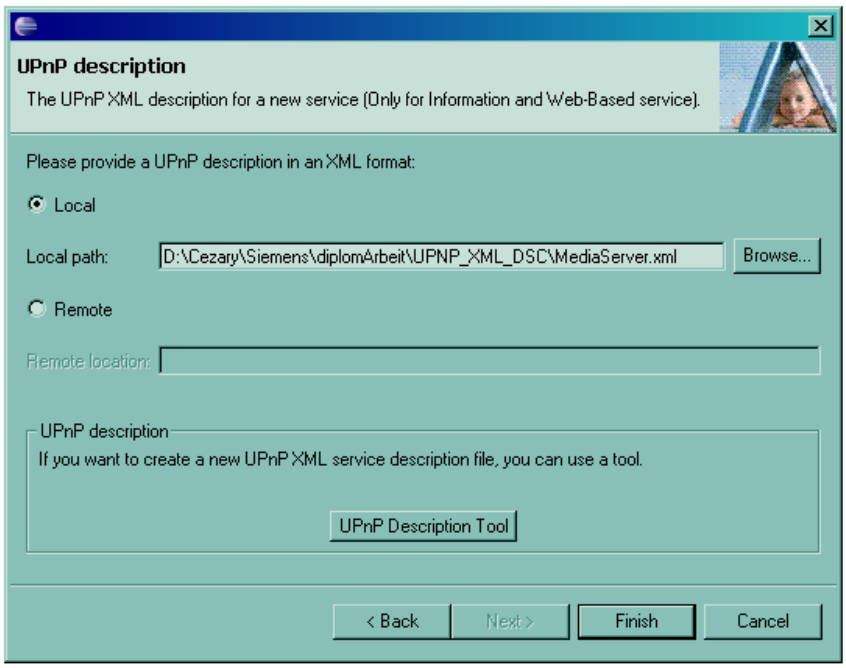
   Please find description of tools and components in Tools_description.doc file

   Hint. The service project under the given name should be generated. The generation of components for OSGi service bundle and Virtual Devices is currently supported. The java files generated are not complete in order to implement a component.



6. On the next page you have to provide UPnP XML description for the implemented service. In the main folder of SSH Service Wizard plugin (where you unpacted all the files) are two folders:
    a. SceneControl
    b. MediaServer

    You can choose to implement one of those two services (choose SceneController.xml or MediaServer.xml file as the device UPnP description).



7. Choose *Finish* to generate the project files

Bibliography

[1] - http://ant.apache.org. Consulted on 23.08.2006.

[2] – www.autosar.org. Consulted on 4.08.2006.

[3] – Manfred Broy. *Challenges in Automotive Software Engineering.*

[4] – Bernd Bruegge, Allen H. Dutoit. *Object - Oriented Software Engineering, Conquering Complex and Changing Systems.* 2000 Prentice Hall.

[5] – Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, Steven Shafer. *EasyLiving: Technologies for I ntelligent Environments.*

[6] - http://www-306.ibm.com/software/rational/offerings/scm.html. Consulted on 22.08.2006.

[7] – Michael H. Coen. *Design Principles for Intelligent Environments.*

[8] - http://www.nongnu.org/cvs. Consulted on 22.08.2006.

[9] – http://research.microsoft.com/easyliving. Consulted on 01.08.2006.

[10] - http://www.eclipse.org. Consulted on 22.08.2006.

[11] - http://www.ethereal.com. Consulted on 22.08.2006.

[12] – www.konnex.org/knx-tools. Consulted on 24.08.2006.

[13] – Martin Fowler, Kendall Scott. *UML Distilled, Second Edition. A Brief Guide to the Standard Object Modeling Language.* 2000 by Addison-Wesley.

[14] - http://www.automation.siemens.com/et/gamma/html_76/products/index.htm. Consulted on 11.08.2006.

[15] - http://www.gstforum.org. Consulted on 6.08.2006.

[16] – M.Hampicke. Seniorengerechte Technik im häuslichen Alltag: Das Smart-Home-Projekt. https://www.sentha.tu-berlin.de/smart/index-smart.html . 2002. Consulted on 26.07.2006.

[17] - http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/overview/index.htm. Consulted on 21.08.2006.

[18] - http://labs.jboss.com/portal/.Consulted on 11.08.2006.

[19] – http://www.cloudgarden.com/jigloo. Consulted on 15.09.2006.

[20] - http://www.ej-technologies.com/products/jprofiler/overview.html. Consulted on 23.08.2006.

[21] - www.junit.org. Consulted on 25.08.2006.

[22] – Michael Kay. *XSLT 2.0 Programmer's Reference Third Edition*. 2004 by Wiley Publishing.

[23] - http://www.knopflerfish.org/eclipse_plugin.html. Consulted on 22.08.2006.

[24] - http://www.konnex.org/. Consulted on 27.07.2006.

[25] – Donald Kossmann, Christian Reichel. *SSL-Running My Web Services on Your WS Platforms*.

[26] – www.livefutura.de. Consulted on 1.08.2006.

[27] - http://logging.apache.org/log4j/docs/index.html. Consulted on 23.08.2006.

[28] - http://logging.apache.org/log4net. Consulted on 23.08.2006.

[29] - http://www.ieclon.com/LonWorks/LonWorksTutorial.html. Consulted on 27.07.2006.

[31] - http://www.magicdraw.com. Consulted on 21.08.2006.

[31] - http://nant.sourceforge.net. Consulted on 23.08.2006.

[32] - www.nunit.org. Consulted on 25.08.2006.

[33] - http://www.omondo.com. Consulted on 21.08.2006.

[34] – www.osgi.org. Consulted on 1.08.2006.

[35] – T.Ostendorf presentation; "Siemens Smart Home. Technical Overview".

[36] - http://www.parasoft.com. Consulted on 23.08.2006.

[37] - http://wiki.ops4j.org/dokuwiki/doku.php?id=pax:logging. Consulted on 23.08.2006.

[38] - http://www.perforce.com. Consulted on 22.08.2006.

[39] - http://www.prosyst.com. Consulted on 22.08.2006.

[40] - http://www-306.ibm.com/software/awdtools/developer/rosexde. Consulted on 21.08.2006.

[41] - http://www.gnu.org/software/rcs/rcs.html. Consulted on 22.08.2006.

[42] – Christian Reichel, Roy Oberhauser. *XML-based Programming Language Modeling: An Approach to Software Engineering.*

[43] - http://www.serve-home.de. Consulted on 8.08.2006.

[44] - http://www.plug-n-play-technologies.com. Consulted on 21.08.2006.

[45] - http://www.hqs.sbt.siemens.com/cctv/content/10/10_010en.htm. Consulted on 8.08.2006.

[46] - http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html. Consulted on 8.08.2006.

[47] - http://www.soapui.org. Consulted on 23.08.2006.

[48] - http://networks.siemens.de/smarthome/de/index.htm. Consulted on 8.08.2006.

[49] - http://subversion.tigris.org. Consulted on 22.08.2006.

[50] – C. Szczegielniak, Markus A. Wischy. *Using FGAG to export data for Visualization.* Konnex conference 2005.

[51] - www.fxl-project.com. Consulted on 15.07.2006.

[52] – Ed Roman, Rima Patel Sriganesh, Gerald Brose. *Mastering Enterprise JavaBeans, Third Edition.* Wiley Publishing, Inc. 2005.

[53] - www.saxonica.com. Consulted on 27.08.2006.

[54] – http://www.golem.de/0502/36040.html. Consulted on 6.08.2006.

[55] - http://www.borland.com/us/products/together/index.html. Consulted on 21.08.2006.

[56] – www.upnp.org

[57] – Dr. Marcus Venzke, “Inter-Object Communication” lecture slides. 27.05.2004 Hamburg-Harburg University of Technology.

[58] - http://www.euproject-victoria.org. Consulted on 6.08.2006.

[59] - http://www.visual-paradigm.com. Consulted on 21.08.2006.

[60] - http://msdn.microsoft.com/vstudio. Consulted on 22.08.2006.

[61] – https://office.microsoft.com/visio. Consulted on 21.08.2006.

[62] - http://www.webinject.org. Consulted on 23.08.2006.

[63] – www.w3.org/2002/ws

[64] – J.Wechsler. *A Software Architecture Selection Method to Simplify Smart Home Service Development.* November 2005.

[65] – Asa MacWilliams presentation; “Smart Home Projects at CT SE 2”.

[66] – Markus A. Wischy; „SURPASS Smart Home. Service Development Guidelines“.

[67] – Andy Wigley, Stephen Wheelwright. *Microsoft .NET Compact Framework*. Microsoft Press. 2003.

[68] - http://www.smarthomeusa.com/info/x10theory/x10theory/#theory. Consulted on 27.07.2006.

[69] – http://www.w3.org/TR/xslt20. Consulted on 7.09.2006.

[70] - http://www.yourkit.com. Consulted on 23.08.2006.