Contract no. 034707 www.mpower-project.eu

m·power

Middleware Platform for eMPOWERing cognitive disabled and elderly

SIXTH FRAMEWORK PROGRAMME: PRIORITY 2.5.11 EINCLUSION SPECIFIC TARGETED RESEARCH OR INNOVATION PROJECT

MPOWER Project Deliverable:

# MPOWER Developers Handbook

Deliverable id:

**D1.2**

| *Key Information from "Description of Work" (from the Contract)* | |
|---|---|
| Deliverable Description | *Developers Handbook. This document will be the main input to developers for implementing applications based on MPOWER platform.* |
| Dissemination Level | PU=Public |
| Deliverable Type | R = Report |
| Original due date (month number/date) | Month 21 / 31.07.08 |
| Release number/date | D1.1/ 22.10.2008 |

| *Authorship Information* | |
|---|---|
| Editor (person/ partner): | Sixto Franco, Greg Loniewski, DI |
| Partners contributing | SINTEF, ARC and UCY |
| Reviewed by (person/ partner) | Marius Mikalsen, SINTEF |
| Checked by and released (person/partner) | Marius Mikalsen, SINTEF |
| Date of release | 22.10.2008 |

# MPOWER Consortium

MPOWER (Contract No. 034707) is a *Specific Targeted Research or Innovation Project (STREP)* within the 6th Framework Programme, Priority 2.5.11 (eInclusion). The consortium members are:

| | |
|---|---|
| **SINTEF ICT (Project Coordinator)**<br>NO-7465 Trondheim, Norway<br><br>Contact person: Marius Mikalsen<br>Email: marius.mikalsen@sintef.no | **Ericsson Nikola Tesla d.d.**<br>Address<br><br>Contact person: Ivan Benc<br>Email: ivan.benc@ericsson.com |
| **Austrian Research Centers GmbH - ARC**<br>Wien, Austria<br><br>Contact person: Barbara Prazak<br>Email: barbara.prazak@arcsmed.at | **Psykiatrien i Vestfold HF**<br>Tønsberg, Norway<br><br>Contact person: Torrid Olathe<br>Email: torhild.holthe@aldringoghelse.no |
| **Uniwersytet Jagillonski Collegium Medicum**<br>Krakow, Poland<br><br>Contact person: Mariusz Duplaga<br>Email: mmduplag@cyf-kr.edu.pl | **TB-Solutions Advanced Technologies S.L.**<br>Zaragoza, Spain<br><br>Contact person: Mayte Hurtado<br>Email: hurtadom@tb-solution.com |
| **University of Cyprus**<br>Nicosia, Cyprus<br><br>Contact person: Eleni Themistokleous<br>Email: eleni.themistokleous@cs.ucy.ac.cy | **Dimension Informatica**<br>Valencia, Spain<br><br>Contact person: Juan Jose Cubillos-Esteve<br>Email: jcubillos@indra.es |

# Table of Contents

# Table of Figures

# 1 Executive summary

## 1.1 Objective

The main goal of the MPOWER project is to create a middleware platform allowing rapid development and deployment of distributed and integrated innovative services addressed to support elderly and cognitively disabled people's everyday activities. The MPOWER middleware services will be applied by software developers for creating their applications. MPOWER offers relevant services to create applications. Software developers must know how to use them as also how to integrate them efficiently, as the main purpose of its usage is speeding up the development process. The MPOWER platform is based on SOA architecture, so by using the MPOWER services application developers reap the benefits brought to them both by the MPOWER platform and the Service Oriented Architecture.

To facilitate the work with the MPOWER platform, this document has been created, serving as a guide for MPOWER-based application developers.

## 1.2 Methods

These developer guidelines has been created based on the experiences gained in the MPOWER project, by the projects own developers. The guidelines are a compilation of the process that we have applied in order to build the two MPOWER applications using the MPOWER platform. As such, these guidelines are built on real life experiences.

## 1.3 Results

Developing new applications from the scratch or using existing components is always easier when having clear guidelines provided. This is what this document intends to provide. It guides developers through the process of using the MPOWER platform, contains guidelines both for application developers and middleware service developers.

## 1.4 Conclusions

The use of web-services is the basic of the MPOWER platform, and knowledge on how to use it is essential. Software developers will not be able to get the maximums benefits from the platform if they do not know how services are used and what you can expect from them. Other MPOWER deliverables explain more in details, the use of services provided by MPOWER platform. This document shows all aspects related to the creation and maintenance of an application or service.

# 2 Introduction

## 2.1 Background and Rational

MPOWER platform helps developers to decrease developing time of applications in the healthcare domain. This document describes how to use the services of MPOWER platform and all benefits one can obtain from using such platform. The use of MPOWER has to be clear and easy in order to really speed up the development process.

Moreover, MPOWER is a platform of services that can be considered as a base for further improvements and extensions. So it is important to get an overview of MPOWER, its architecture, general ideas, maintenance details and way of its enhancement.

## 2.2 Role of this deliverable

The main purpose of this document is to assist developers in using the MPOWER middleware platform when facing the task of building applications intended to be used by elderly and cognitive disabled people. Using MPOWER platform as a base avoids starting the implementation from scratch and thus saves developers time.

Likewise this document includes guidelines for developers to improve MPOWER by adding new services to the platform. The document provides an example of how to create and deploy a MPOWER service in a step by step guide. It also describes all necessary MPOWER issues that an application developer could need, two aspects of MPOWER services (how they are implemented and how they can be used), application program interfaces (APIs), toolchain (used by MPOWER developers - not mandatory) and some case studies. Finally, the document includes a step by step guide of how to develop an application based on MPOWER platform.

It is important to differentiate between the application developer and the middleware developer. The application developer will use this document to:

- Get information on the use of MPOWER middleware services in an application development.
- Understand which type of systems that can be built using the MPOWER middleware.
- Understand the structure and usage of MPOWER.
- Know APIs of the services provided in MPOWER.

The middleware developer will use the document to:

- Understand how the MPOWER middleware is designed.
- Guide through implementation and maintenance of the MPOWER middleware.

## 2.3 Target audience

Two principal target audiences are identified: application developers creating applications based on MPOWER (using services and facilities offered by MPOWER) and developers who will be in charge of maintaining and enhancing the MPOWER platform. In this document there is a guide for both groups of developers.

It is recommended that the readers and users are familiar with web service development. If you are not, we recommend the following resources to support you in the use of this development guide:

- Java web service resource [1]

- Web service resource [2]

## 2.4 Relationship to other MPOWER deliverables

- **D1.1 – Overall architecture**: Presents the project approach and reference architecture. The architecture will impose functional requirements on each of the middleware blocks.

- **D1.3 – Service Lifecycle Model**: Based on the overall architecture of MPOWER platform, functional description is provided in a form usable for application developers.

- **D2.2:** This document describes in detail the design and implementation of components and services of middleware for the sensor and smart house management.

- **D3.1:** This document describes in detail the design and implementation of the **medical and social information** middleware providing information on the components and services of the **medical and social information** middleware.

- **D3.2:** This document describes in detail the design and implementation of the **context component which manages the context information**.

- **D4.2:** This document describes in detail the design and implementation of the **interoperability** middleware services.

- **D5.2:** This document describes in detail the design and implementation of the **security** middleware services.

- **D6.2:** These documents describe case studies based on the experience acquired during the Proof of Concept Applications (PoCA's) development phase. These PoCA's use services provided by MPOWER platform and the knowledge generated during their development process.

## 2.5 Structure of this document

This document consists of seven chapters, each of which deals with different aspects that application developer using MPOWER platform has take into account when creating a new application.

The first chapter includes an executive summary. The current chapter provides a brief overview of the role and structure of the document, and the relationship to other MPOWER deliverables.

The third chapter demonstrates the MPOWER architecture with a short Service Oriented Architecture (SOA) description. That section starts introducing basic concepts and goals of the platform. It continues providing information on its SOA reference architecture, the components forming the platform and their correlations. Profiles of platform users are also identified and described here.

In chapter 4, the installation of MPOWER platform is described. After reading this chapter, the developer should know the necessary factors to set up MPOWER platform and what are the actions to be performed when using MPOWER.

Chapter 5 describes how developers can create an application based on MPOWER. Moreover all services that are available in MPOWER platform are listed.

Chapter 6 illustrates how a new service can be created and added to MPOWER platform. With such knowledge MPOWER can be easily upgraded.

# 3 Description of the MPOWER platform

## 3.1 Introduction

MPOWER is an open platform to simplify and speed up the process of developing services for people with cognitive disabilities and elderly. This platform provides reusable, flexible, and interoperable service specifications and implementations. Services offered are simple, general and can be adopted in a custom manner (by application developer) to deal with any kind of scenario that can be detected. The platform in particular supports integration of SMART HOUSE and sensor technology, interoperability between domain specific systems, secure information transfer and its management, including both social and medical information; and mobile users which often change context and tools. MPOWER aims to take into consideration the main issues related to care of elderly and cognitive disability person which can be supported by ICT solutions.
Basically the main idea of MPOWER is to give the application developer a reusable base of services designs and components/services, avoiding starting from scratch, saving developer's time and allowing the developer to build application from a set of small pieces which all together create the complete application. With reference to Figure 1, the reusable artefacts from MPOWER are :

- UML Specifications of Service defined as both Platform Independent Models (PIM) and Platform Specific Models (PSM)
- Domain Specific UML extensions, including UML profiles for Homecare to enable specification of domain
- The MPOWER toolchain enabling development of domain services. The toolchain uses the UML extension and reference architecture to improve model transformation and code generation
- Runnable Domain Specific Services: the MPOWER Middleware Services that are developed and available as deployable archives.
- Methodology for developing services and applications in the domain

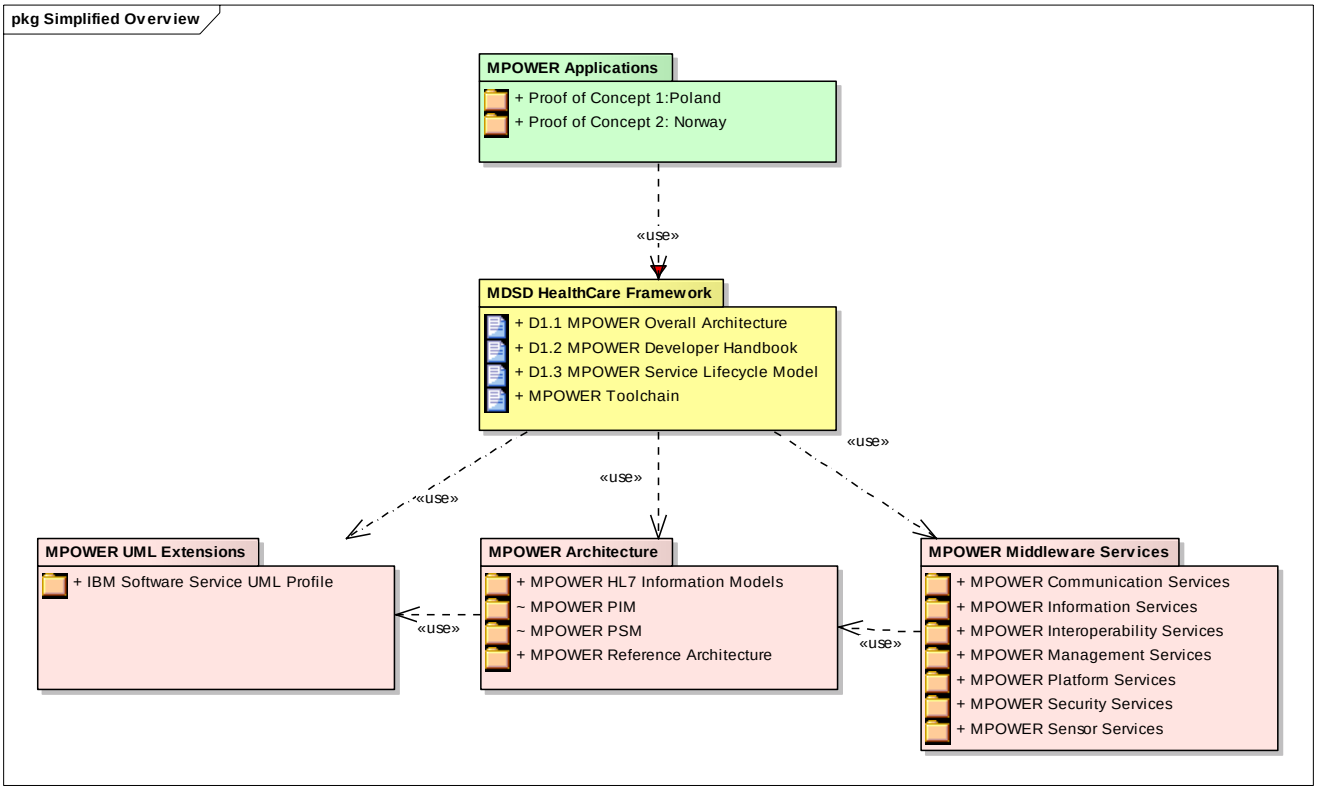


**Figure 1: Overview of artefacts and their relationships in MPOWER**

## 3.2 Who should use the MPOWER Platform

MPOWER is addressed to developers in the healthcare domain especially those who create applications for cognitive disabled and elderly people.

MPOWER is a platform helping in design, development and deployment of interoperable applications offering innovative end-user services.

There is potentially a large market for healthcare domain services related to aforementioned cognitive disabled, elderly people. Also nowadays the scope of innovation seems to be attractive. On the other hand technical difficulties and resulting high development costs make it commercially unattractive. MPOWER overcomes these difficulties by:

- specifying and implementing components which solve common technical problems when creating new services,
- a dynamic mapping between medical, social and context information models, sensor and legacy system interfaces (Information Services),
- challenging the problem of combining SMART HOUSE technology and task specific applications used by different stakeholders within interoperable services (Sensor Services),
- providing structured mechanisms for representing and adapting to changes in user context in a distributed or mobile environment (Context Manager),
- promoting standardisation by aligning the M•POWER approach which supports the HL7 and promoting the platform for members of EAHSA ,
- applying an SOA (Service Oriented Architecture) based architecture that relies on service-orientation as its design principle. Service-orientation describes an architecture that uses loosely coupled services to match up the user's requirements and also requirements of business processes. Another advantage is avoiding code duplication.

## 3.3 MPOWER Reference Architecture

Reference architecture is defined as:

*"A high-level, generic architecture which is used as the basis for development of concrete system architectures, and to compare architectures of existing systems to each other"*

The main purpose of using common reference architecture in MPOWER is to aid developers (application and middleware) in the definition of services; and also to provide a common and unique abstraction level and concepts for interaction.

The MPOWER SOA reference architecture [3] is based on IBM's [4] reference architecture for Service-Oriented Architecture which consists of five layers. Each layer comprising a set of "components" that conforms to the rules and requirements specified for the layer. Furthermore, the layers are grouped into three groups: application, domain and system specific group. Such grouping is done to clearly separate the components that are specific for the applications, for the domain and underlying systems. The figure below shows a general view of MPOWER architecture following the IBM SOA reference architecture.

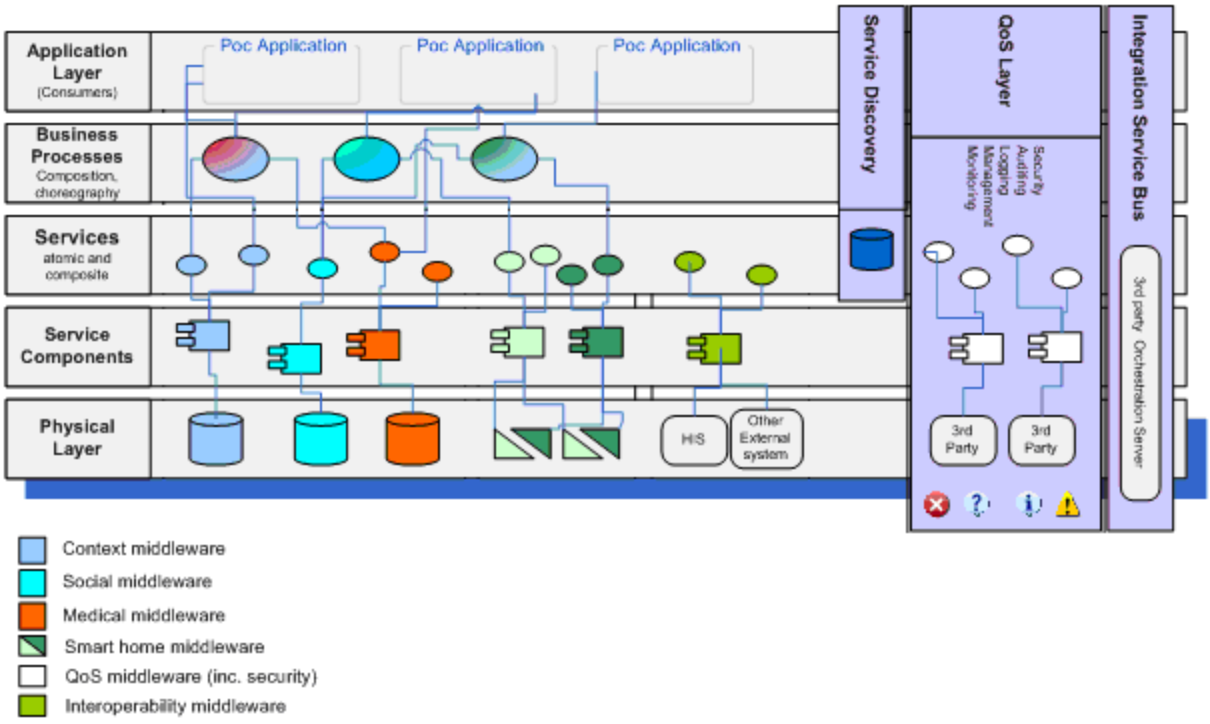


**Figure 2: MPOWER Reference Architecture**

D1.1 "Overall Architecture" [3] document provides a description of all architectural aspects of an MPOWER based information system. Please refer to it to know more on the MPOWER reference architecture.

# 4 Installation guide

## 4.1 Required tools, middleware and libraries

This subchapter describes the set of tools, middleware and libraries which are required in different scenarios when developing, testing and deploying software based on MPOWER.

The MPOWER middleware is based on a set of technology standard for which there exist multiple implementations and multiple tools. The standards applied are:

- UML 2.1
- Java 2 Enterprise Edition (J2EE), version 5.5 or later
- JAX-WS 2.0 (JSR 224) for web services (and not the predecessor JAX-RPC)
- Standard SQL based database

Beyond these standards, some further tools and libraries have also been found to be useful during the development of the MPOWER middleware, services, and proof-of-concept applications. These are also listed here, but are optional. Some of these tools are also used in later sections of this handbook, e.g. in the step-by-step guides for developing services and applications.

The MPOWER middleware is implemented in Java, and a Java runtime environment is needed in order to deploy the middleware and services. Although future services extending the MPOWER platform could also be developed in other technologies (e.g. .NET), developing such services in Java allows the services to be deployed to the same application server as the initial MPOWER services.

Some of the installations described here require that you have administrator rights to the computer on which to install.

### *4.1.1 Overview of requirements for scenarios*

Each of the following subsections gives a summary of the requirements for a development / testing scenario. In each of these, only a list of the required tools, middleware and libraries are given. The ordering of the entries in these lists is also the recommended order of installation. Installation instructions and version information for each entry is in these lists can be found later in this chapter.

| **Developing information services and applications** | |
|---|---|
| UML Modelling | Required |
| Java Development Kit (JDK) | Required |
| Java IDE | Required |
| Object/relational persistence service | Optional |
| Database driver libraries | Optional |

| **Deploying and testing information services and applications** | |
|---|---|
| Java Development Kit (JDK) | Required |

| | |
|---|---|
| Application Server | Required |
| Object/relational persistence service | Required |
| Database | Required |
| Database driver libraries | Required |
| Web service testing tool | Optional |

| **Developing physical level services** | |
|---|---|
| UML Modelling | Required |
| Java Development Kit (JDK) | Required |
| Java IDE | Required |
| Sensor's specifications | Required |

| **Deploying and testing physical level services** | |
|---|---|
| Java Development Kit (JDK) | Required |
| Application Server | Required |

### *4.1.2 Tool, middleware and library details*

#### 4.1.2.1 UML Modelling

A central tool in the MPOWER service development tool-chain is the modelling tool. The support for modelling and transformation developed in the project is for the tool Enterprise Architect.

***Recommended tool: Enterprise Architect***

***Version: 6.5 or higher, professional or corporate edition.***

***Note that at the time of writing, the 7.x versions of this tool has a bug which prevents correct WSDL generation, and thus causes some problems in the MPOWER toolchain.***

***The tool can be purchased from Sparx Systems:***

***http://www.sparxsystems.com.au/***

To install the tool, follow the instructions provided with the tool.

**Installing the UML Profile for Software Services in Enterprise Architect**

The recommended approach for modelling of MPOWER applications is based on IBM's UML Profile for Software Services. This profile is described at:

http://www.ibm.com/developerworks/rational/library/05/419_soa/

The profile is supported by a RSA Plug-In from IBM, which can be downloaded from:

http://www-128.ibm.com/developerworks/rational/library/05/510_svc/

To convert this to a UML profile usable by Enterprise Architect, the following steps are needed:

1. unzip the downloaded file (softsvc.zip)
2. unzip the file com.ibm.rational.softsvc_6.0.0.1.jar found in the plugins directory from step 1
3. rename the file SoftwareServices.epx from from .epx to .emx
4. from Enterprise Architect, first create a new project which will contain the profile
5. select “Import Model from XMI” and click the “Import EMX / UML2 Files” button, then select the SoftwareServices.emx file
6. select “Save Package as UML Profile” to save the resulting package as a uml profile suitable for use with Enterprise Architect

The new profile can now be imported in the Enterprise Architect projects in which it will be used by selecting “Import Profile” in the Resources view.

**Installing the WSDL transform in EA**

The MPOWER toolchain contains a customized template for WSDL transformation called MPowerWSDLTemplate.xml. This template can be downloaded from the MPOWER web site. To use the template, it first has to be imported into the Enterprise Architect project. This is done by choosing the "Import reference data" from the "Tools" tab. This import will modify some of the built-in transformations for WSDL.

#### 4.1.2.2 Java Development Kit (JDK)

For developing services and applications in Java, a Java development kit is needed. Some development environment comes with the JDK bundled, otherwise it should be installed before other tools.

***Version: JDK 6.0 or higher (required)***

***Can be downloaded from:***

***http://java.sun.com/javase/downloads/index.jsp***

#### 4.1.2.3 Java IDE

The choice of a Java integrated development environment (IDE) is open to the developer, as MPOWER do not require the use of a particular one. The environment should however include support for J2EE and JAX-WS 2.0, and preferably have built in facilities or extensions which allow the developer to import WSDL definitions and generate skeletons for web service implementations based on these.

Netbeans is an example of a suitable Java integrated development environment (IDE), and is also used in examples provided in later parts of this document. Eclipse is another alternative, but no examples of usage for this are provided.

***Recommended tool: Netbeans***

***Version: NetBeans 6.0 or higher***

***Netbeans can be downloaded from:***

***http://www.NetBeans.org/index.html***

***We recommend a download which includes support for at least Base IDE, Java SE, Web & Java EE, SOA, and GlassFish.***

#### 4.1.2.4 Object/relational persistence service

Hibernate is an object/relational persistence service which allow development of persistent classes using object-oriented mechanisms. Hibernate was used in the implementation of some of the predefined services of the MPOWER platform. When developing new services, the use of Hibernate is optional. However, the Hibernate libraries are needed to deploy and test the predefined MPOWER services.

***Recommended tool: Hibernate***

***Version: Hibernate Core 3.2.4.SP1***

***Hibernate can be downloaded from the following location, and information of how to install it can be found in the on-line documentation at the same site:***

***http://www.hibernate.org/6.html***

***The "setting up you environment" section of the following link provides some information of how to install Hibernate in the Netbeans environment.***

***http://www.NetBeans.org/kb/articles/hibernate-javaee.html***

***Please note that other sections of the same page seem to be somewhat outdated (e.g., the "Runtime" window has been renamed to "Services" in Netbeans 6.0, and some wizards are no longer available).***

#### 4.1.2.5 Database

Developers of new services for MPOWER can make their own choice of database. The code for the MPOWER middleware has been developed with database-neutrality in mind. However, the middleware has only been tested using an Oracle database, so the degree of compatibility with other databases is unknown.

***Recommended tool: Oracle***

***Version: Oracle 10g Express edition (XE)***

***Can be downloaded from:***

***http://www.oracle.com/technology/xe/index.html***

***Follow the install instructions provided from Oracle.***

#### 4.1.2.6 Database driver libraries

To use a database from an IDE, e.g. the Netbeans environment, you will need a driver library for the database. The following describes the installation of the required database libraries, e.g. Oracle libraries, in Netbeans:

Open the Services window of Netbeans, and select Drivers under Databases. Select “New Driver…” from the pop-up menu, click “Add…” in the dialog box, and the select the driver file.

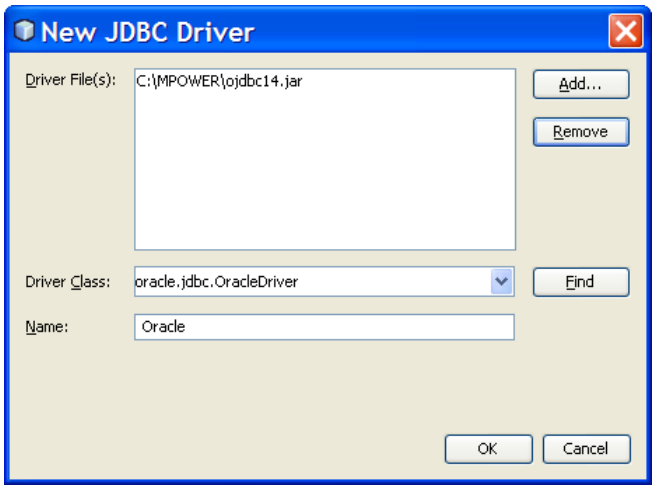


**Figure 3: Adding the JDBC driver to Netbeans**

To connect to an existing database, select: the new driver that appear (e.g. Oracle), and click “Connect using…”. A dialog will appear in which you should fill in the URL of the database to connect to, along with user name and password.

***Recommended tool: Oracle JDBC driver***

***Version: Oracle 10g JDBC driver for JDK 1.4***

***Can be downloaded from:***

***http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html***

***At the time of writing, the name of the file to download is ojdbc14.jar, and is found under the Oracle 10g Release 2.***

#### 4.1.2.7 Application Server:

To deploy and test the MPOWER middleware and new services and applications based on it, a J2EE application server is needed. The MPOWER middleware was developed and tested using Glassfish.

***Recommended tool: Glassfish with SOA add-ons***

***Version: Glassfish v2***

***Download installation from one of the following sites:***

- ***i) https://glassfish.dev.java.net/downloads/v2ur1-b09d.html***
- ***ii) https://open-esb.dev.java.net/Downloads_OpenESB_Addons_NB6.html***

***For development purposes, you may instead use the Glassfish distribution which is bundled with some of the Netbeans installations (see the Netbeans section).***

#### 4.1.2.8 Web service testing tool

soapUI is an open source web service testing tool. The tool exists in several variants, e.g. as plug-in for Netbeans and Eclipse. As Netbeans is the recommended IDE in MPOWER, the references to the soapUI tool in this document refer to the Netbeans plug-in version.

***Recommended tool: soapUI Netbeans Plugin***

***Version: soapUI 2.0***

***See downloaded and install instructions at:***

***http://www.soapui.org/Netbeans/installation.html***

***The tool is also bundled with the OpenESB add-ons for Netbeans from the Netbeans version 6.0 M9 / OpenESB 2.0.***

#### 4.1.2.9 Enterprise Service Bus

Enterprise Service Bus is an integration environment based on JAVA Business Integration (JBI). It provides an abstraction layer on top of the messaging mechanisms and moreover a mechanism that makes use of the business processes concept which plays the role of controllers for the messages flow. This concept is closely related to the software architecture, especially in the middleware SOA infrastructures. ESB allows integrate web services and enterprise applications defining the software infrastructure.

The ESB environment works based on the following concepts: Service Engines (SE), Binding Components (BC) and a Normalized Message Router (NMR). ESB extends the JBI with additional services engines performing certain business logic, binding components for different messages formats, as well as tools and services for managing and monitoring the environment.

MPOWER project uses OpenESB which is SUN's implementation of ESB. By now it does not work as a separate runtime environment, but is integrated with the Glassfish application server and Netbeans IDE. MPOWER platform architecture is planned to use the bpel-service-engines from the OpenESB implementation (in the case of notification business process) as also the ESB as the messages interchanging medium (in the case of the FSA and ContextManager component).

***Recommended tool: OpenESB***

***Version: Oracle 10g Express edition (XE)***

***More information can be found in:***

***https://open-esb.dev.java.net***

***Can be downloaded together with Netbeans IDE from:***

***https://open-esb.dev.java.net/Downloads.html***

#### 4.1.2.10 Rules Engine

A rule engine is a computer program that tries to derive answers from a knowledge base. It is the "brain" that expert systems use to reason about the information in the knowledge base for the ultimate purpose of formulating new conclusions. Rules consist of two parts: a sensory precondition (or "IF" statement) and an action (or "THEN"). If a production's precondition matches the current state of the world, then the production is said to be triggered. If a production's action is executed, it is said to have fired.

Rule engine in MPOWER is based on JBoss Rules, an open source and standards-based business rules engine. MPOWER uses the rule engine for guessing whether a context change has happened, and in positive case give an answer to new situation.

***Recommended tool: JBoss Rules***

***Version: JBoss Rules 4.0.0.12865***

***It can be downloaded from:***

***http://www.jboss.com/downloads/index***

## 4.2 Getting access to the MPOWER Middleware and Tools

The MPOWER Middleware and Tools will be available for download. At the time of writing, the hosting location has yet to be decided. The hosting location will be published on the project web site:

http://sourceforge.net/projects/mpower

# 5 Application Developer's guide

## 5.1 Introduction

From the Application Developer's perspective the MPOWER provides loosely-coupled sets of functionalities – services, which can be easily integrated into meaningful applications.

Figure 4 identifies the application developer role in the several roles that exist in the design, implementation and use of the MPOWER platform.

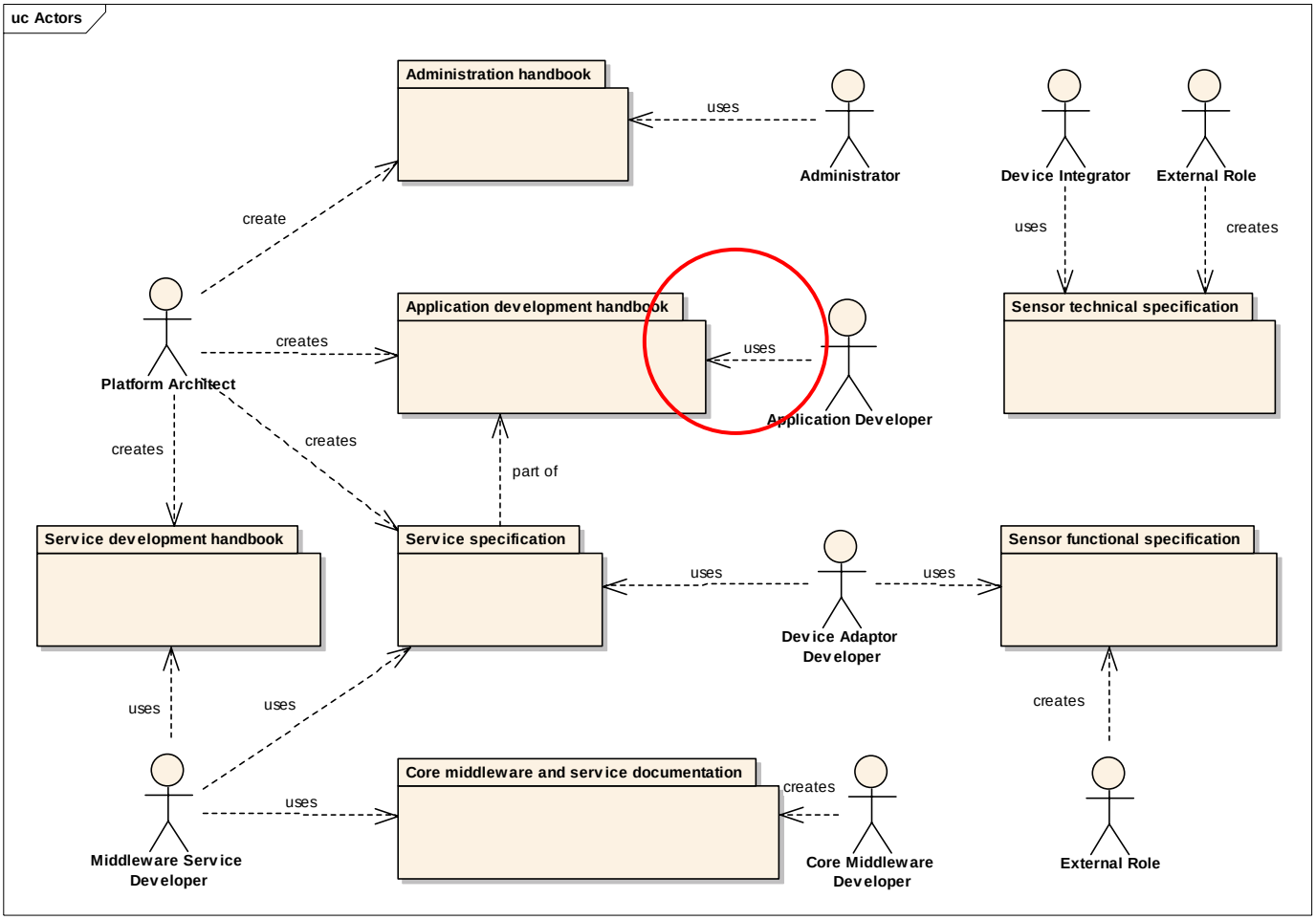


**Figure 4: The Application Developer in the MPOWER Actor relationship**

Preferential the Application developer should use the MPOWER middleware to build applications this includes in detail the following (see Figure 5):

- To create and design a graphical user interface (GUI). The user interface is the communication interface for the user which are the care givers and the person of care. Relating to the stakeholders there are also needs for special aspects in the design of the GUI which have to be fulfilled.
- The Application developer has also to build business services by using choreography. The application developer has to call the service directly. Therefore he has to screen the service specification to find the functionality needed. For different business functionalities it is also

possible to combine the functionality of certain services. A certain business bus has been implemented. It is the choreography bus to inform different applications and modules. It's the Notification Bus (explained in detail in deliverable D4.2. If there is a need on other business functionalities of the application the application developer has to use choreographing mechanisms like the Enterprise Service Bus [5] and different services to build business service functionality. But this business choreography development is normally done by the middleware developer.

- Building application includes also to instantiate the existing middleware services depending the specific dedication this could include to adapt the services to the behaviour for specific users

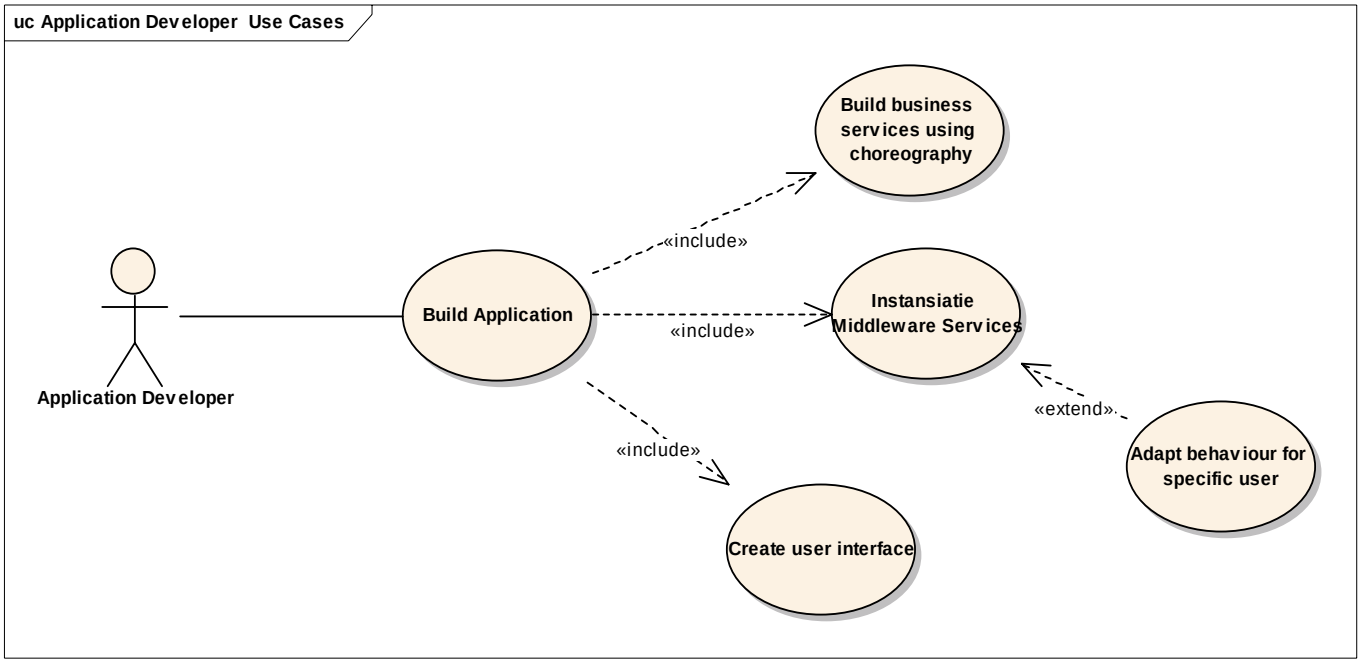


**Figure 5: Use Case of the Application Developer**

## 5.2 MPOWER services

The MPOWER project follows the SOA approach. The MPOWER Middleware is a collection of reusable services and components that can be categorized in five different groups of services. The categories are:

- **Management Services:** services for managing services, users, access rights and system contexts.
- **Information Services:** services setting and getting information for individual plan, calendar, medication list, and knowledge sources.
- **Sensor Services:** services for configuring (add, remove, adjust) devices and retrieving sensor information.
- **Security Services:** services for authentication and authorization of users and system components.
- **Communication Services:** services for sending messages and notifications to users and systems.

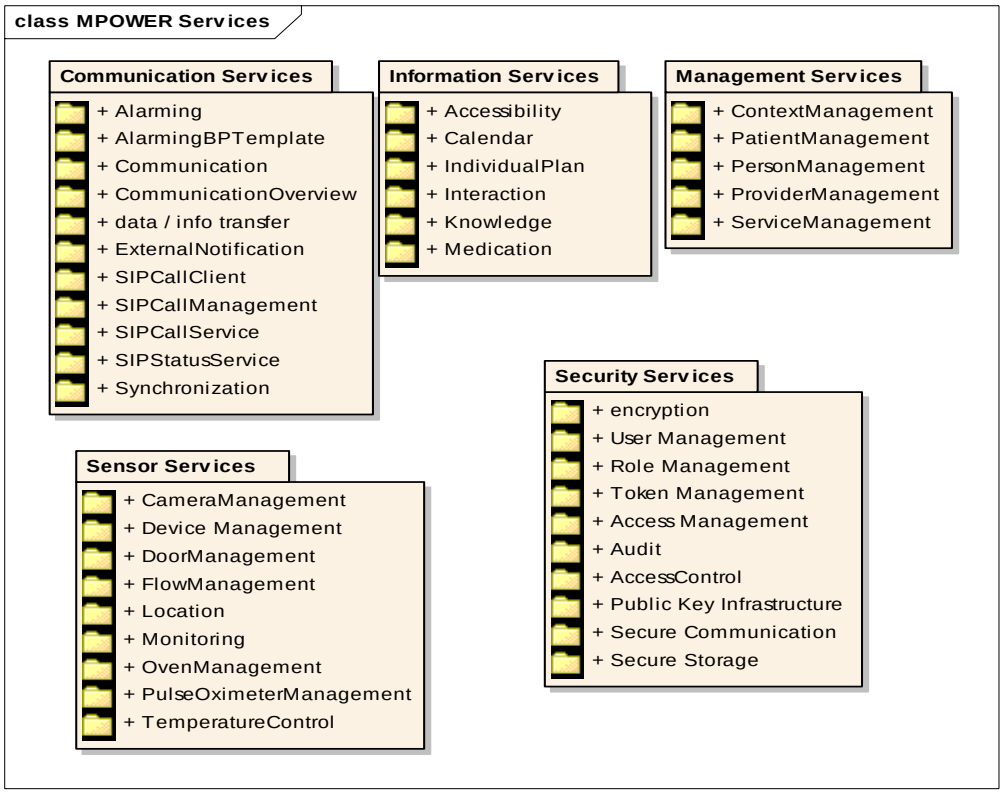


**Figure 6: The MPOWER services**

### *5.2.1 Communication Services*

The Communication Services are mostly services dealing with interoperability features inside the middleware as well as to services and functionalities outside the MPOWER middleware.

In **Error! Reference source not found.** you can see the overview of the communication services from the Enterprise Architect Model. The communication services are clustered in services dealing with synchronization requirements, data / info transfer, external notification, alarming and communication in general.

The synchronization services as well the data / info transfer services are coming from a requirement to communicate data from the internal middleware to external system. These are in general medical content data of the users which should be provided over an interface to any legacy systems. This requirement should show that the middleware is no proprietary systems and there are open interfaces.

For demonstration this requirement is solved with the Calendar and Medication export services. The description of this services can be found in the “D4.2 – Interoperability Middleware Design” deliverable. These services provide a export of medical data in the Continuity of Care Record (CCR) format as well as the ICal format.

The Journal Note transfer was as a requirement which was not prioritised during the project but the functionality to implement it is similar to the design of the services explained in the D4.2 deliverable.

The external notification service are also explained and implemented in the WP4 work package and the D4.2 deliverable. This service provides an interface where other services can send notification messages (SMS and Email) with a special content to a certain recipient. This functionality will be used for notification as a reminder or some low important alarming cases.

The services for the alarming functionality are integrated with a so called notification bus. These functionalities together provide a alarming handling and a locking of alarming events as well as a bus

system for informing and handling the request and the notifications of different client and application modules. These functionalities are also explained in the D4.2 deliverable.

All communication requirements are solved with the SIP based Voice / Video Communication Services. These services are able to provide a voice and video communication between different client stations but also a voice communication to outside external phones. The design and the description of these services are also to find in the D4.2 deliverable.

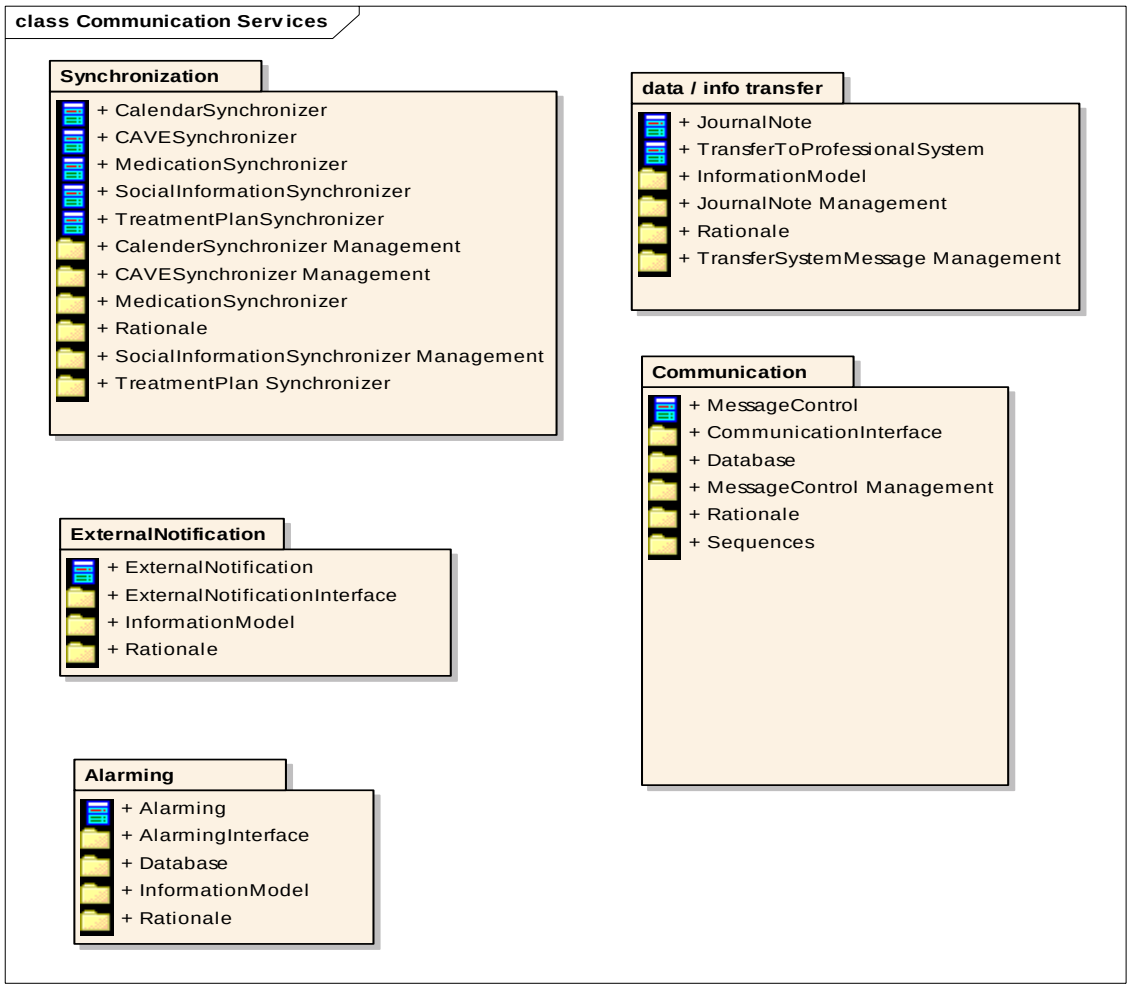


**Figure 7: Communication Services**

## *5.2.2 Information Services*

The information Services are dealing with the internal management of the information of the users. They are basically the services implemented and tested in the Norwegian POCA. Some of the requirements for services you see in **Error! Reference source not found.** have been implemented some other have finally not prioritised enough.

The services that have been implemented are the services in the medication package. All the information management inside the middleware is based on the HL7 format. Different stakeholders are able to subscribe medications to the user via the web interface. The users on his application are able to have always a overview of which medication to take. The medication content is stored in the HL7 structure inside the middleware database.

Also the service in the Calendar and Individual Plan package has been implemented. They manage a personal calendar and different appointments as well as different contacts.

The design and the service description of the implemented Information services are explained in the "D3.2 – Design middleware service for context information" deliverable.

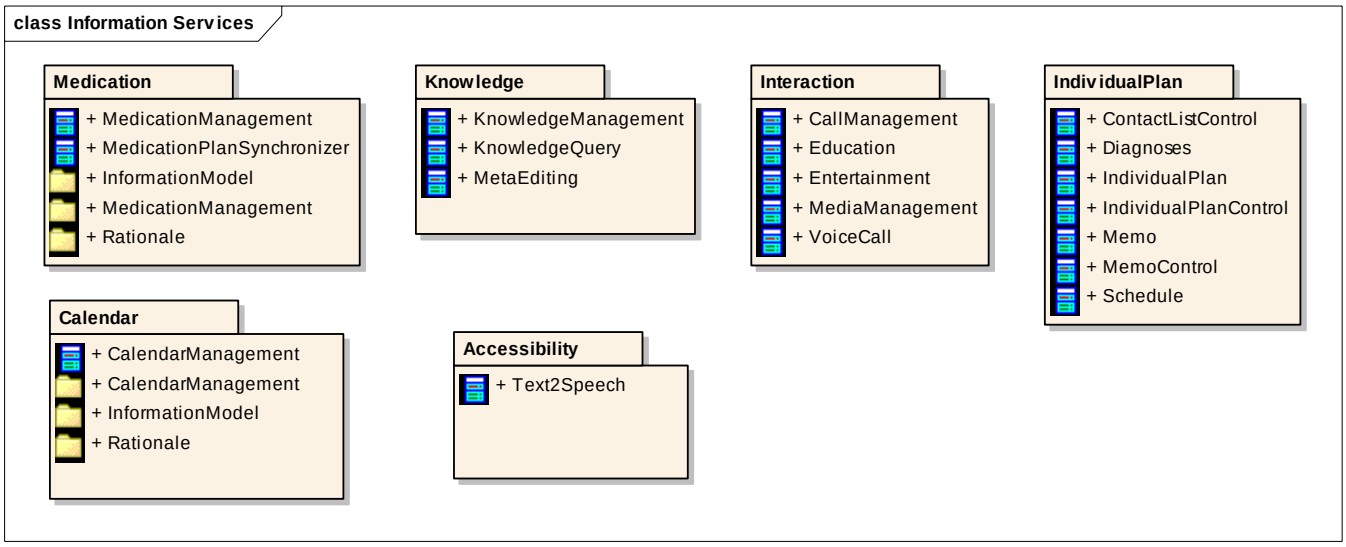


**Figure 8: Information Services**

### *5.2.3 Management Services*

The Management Services are implemented in different in different work packages.

Concerning the Service Management requirements there has been implemented a UDDI Service Registry. The design and the description of the UDDI Service Registry can be found in the D4.2 deliverable.

The Context Management is part of the Sensor Services because it managed the context which is mostly detected by different sensors. The Context Management is explained in the D2.2 deliverable.

The service functionality in the Actor Management cluster is implemented together with the security service, because the actor management is strong connected with the rights of the different stakeholders. All information concerning certain stakeholders and their user rights are stored in the MPOWER database and should be defined and configured when setting up the environment for a certain user.

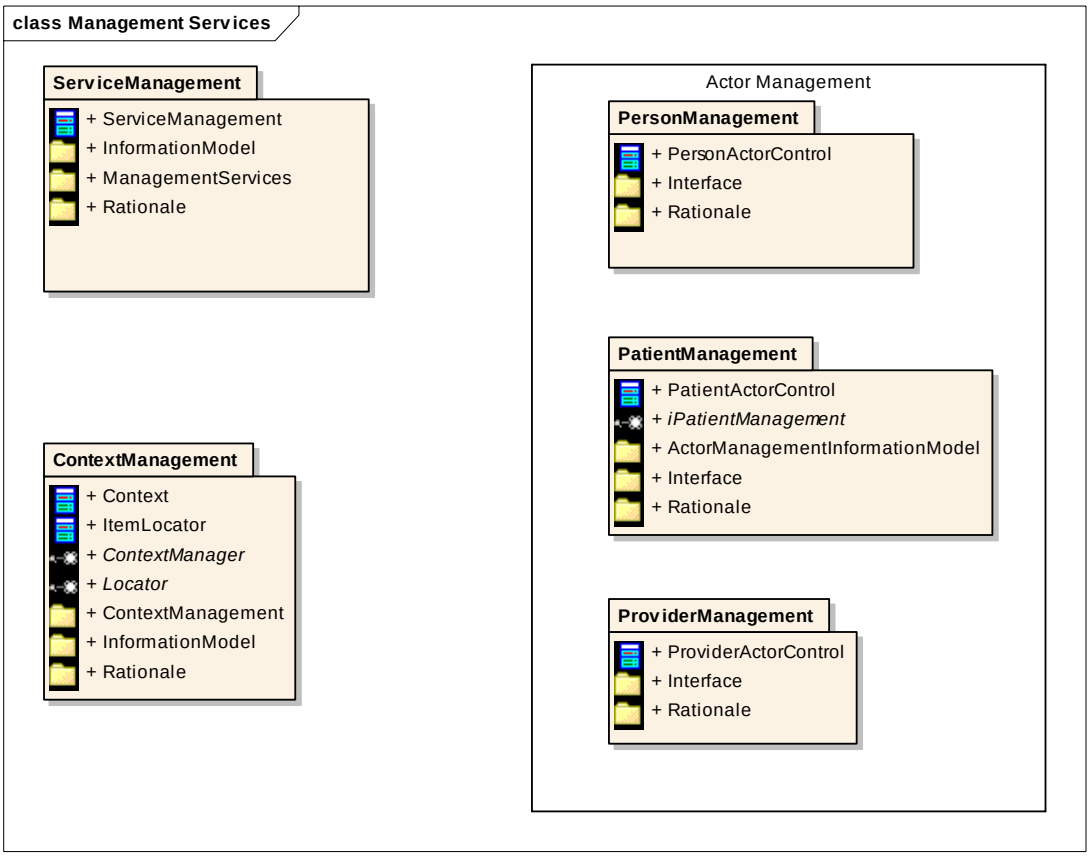


**Figure 9: Management Services**

### *5.2.4 Security Services*

The Security Services have been implemented as a feature which provides security in all middleware layers. There are different kinds of security functionalities for services and actors. Basically different actors are identified using tokens which allow or deny the permission in a particular case. Not every actor can use all services. The restriction is assigned to the role which the actor possesses in the use case (e.g. patient and therapist).

A detailed explanation of the different security functionalities you can find in the “D5.2 – Design of Security Middleware”.

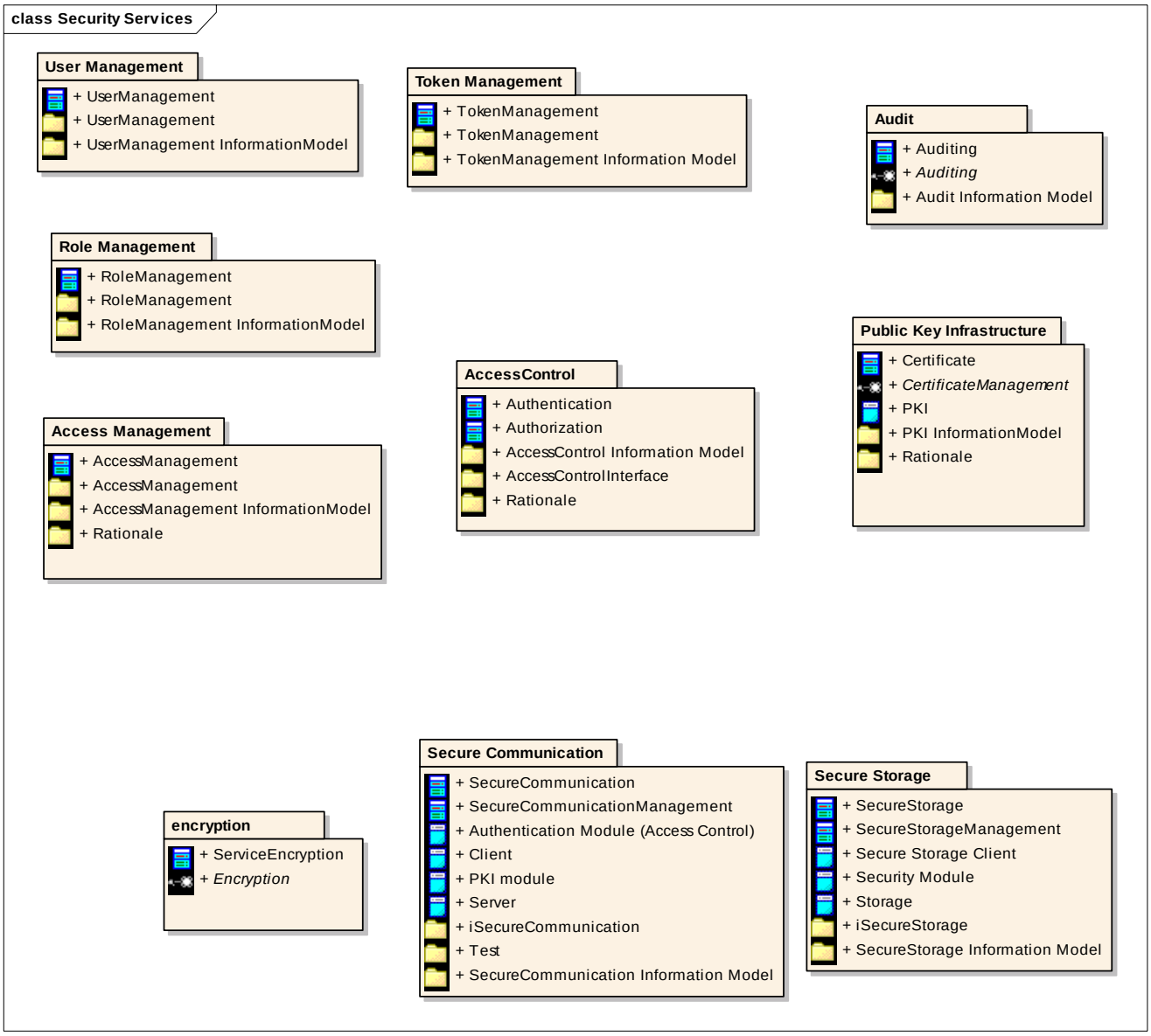


**Figure 10: Security Services**

### *5.2.5 Sensor Services*

The whole variety of sensor services providing the functionality on the physical layer. This means all the hardware is connected with the functionality of these services. For demonstration and for fulfilling the use cases different hardware sensors and different sensor services has been implemented including services for door control, for temperature control, for location control etc.

For providing an interoperable layer where different sensor from different providers can be connected to the FSA (Frame Sensor Adapter) has been introduced.

The detailed information about the FSA and the sensor services can be found in the “D2.2 – sensor service middleware design” deliverable.

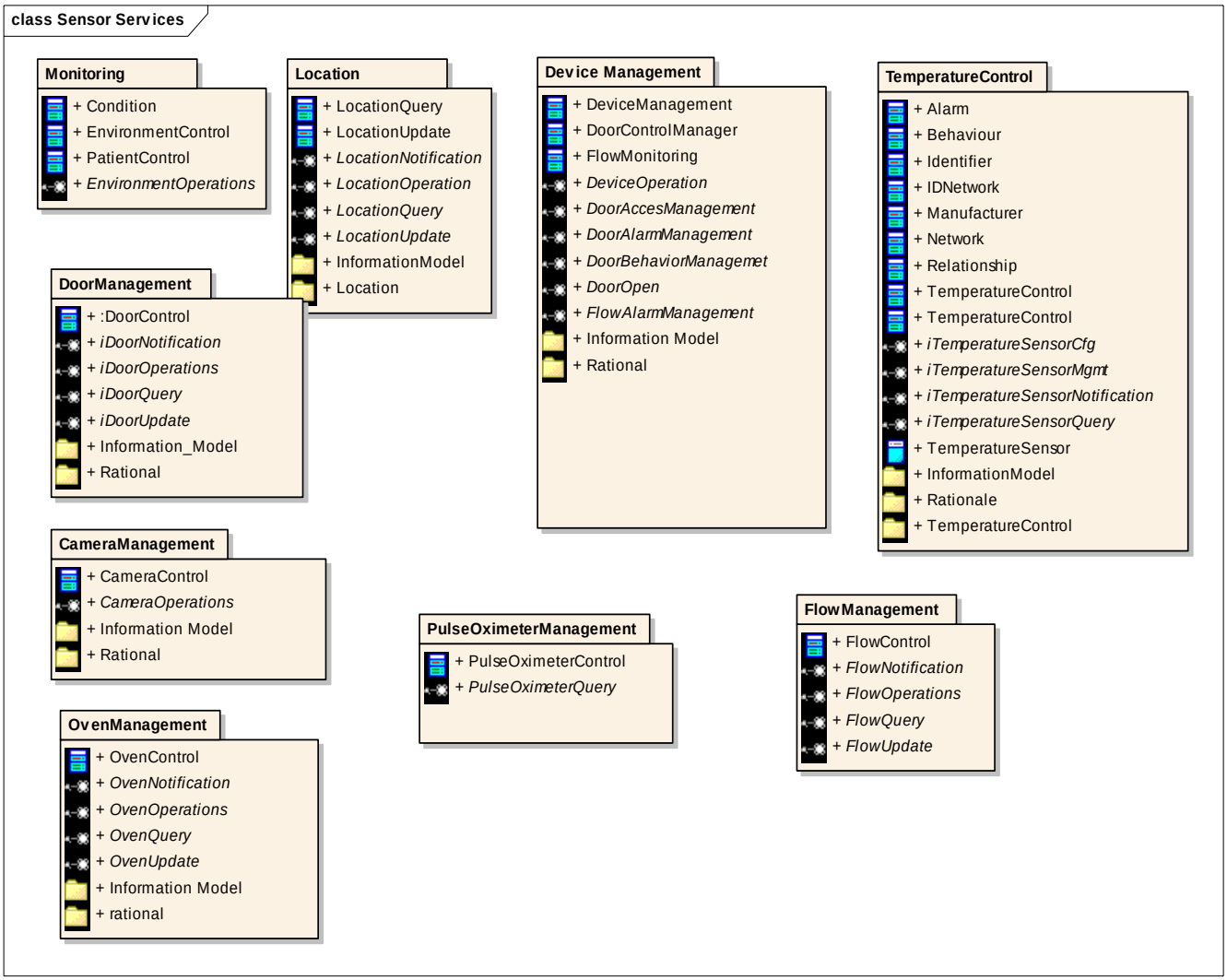


**Figure 11: Sensor Services**

### *5.2.6 MPOWER components*

MPOWER middleware offers a set of services providing functionality to develop applications regarding care issues. The visible parts for application developer are services that can be used to create aforementioned application. Those services are divided into the following groups: security services, communication and interoperability services, medical and social information services, as also sensor services. Moreover MPOWER offers other components that are not associated directly with any of those groups of services. Most of the MPOWER services are accessible via web services.

What is called as “MPOWER component” is that part of MPOWER platform that is not offered as a service but runs inside the platform carrying out different kinds of tasks important for combining other parts of the platform and securing its proper usage. Some of these MPOWER components are: ESB (Enterprise Service Bus) [5], Rules Engine [6], alarming, data bases and FSA [7]. There no exists a direct use of these components by developers, but they can carry out important task to get well performance of application. As they are not service offered by MPOWER platform, they are not explained in service explanation document. However they need to be explained to developers for knowing how they work and how the can be used. Hence, next section will describe the, because they have not a document where are described, as service have.

**Enterprise Service Bus**

Service-oriented architecture (SOA) needs an infrastructure that can connect any IT resource regardless its technology and the place of its deployment. To be flexible it needs an infrastructure that can easily combine and re-assemble services to meet changing requirements without disruption. To be dependable it needs an infrastructure that is robust and secure. This infrastructure is the enterprise service bus (ESB).

The word "bus" is a reference to the physical bus that carries bits between devices in a computer. The enterprise service bus has an analogous function on a higher level of abstraction. Making use of an ESB, an application can communicate via the bus which acts as a message broker between applications. The primary advantage of such approach is that it reduces the number of point-to-point connections required to allow applications communication. This makes analysis for major software changes simpler and more straightforward. By reducing the number of points-of-contact to a particular application the process of adapting a system to changes coming from one of its components becomes easier.

The ESB is the piece of software that lies between the business applications and enables communication among them. Ideally, the ESB should be able to replace all direct contact between applications by passing it through the bus. In order to achieve this objective the bus must encapsulate the functionality offered by its component applications in a meaningful way. This is typically accomplished through the use of an enterprise message model. The message model defines a standard set of messages that the ESB will both transmit and receive. When it receives a message, it routes it to the appropriate application. Therefore the ESB is in charge of carrying out the internal communication (between different components of the platform) and external communication (between the platform and other applications).

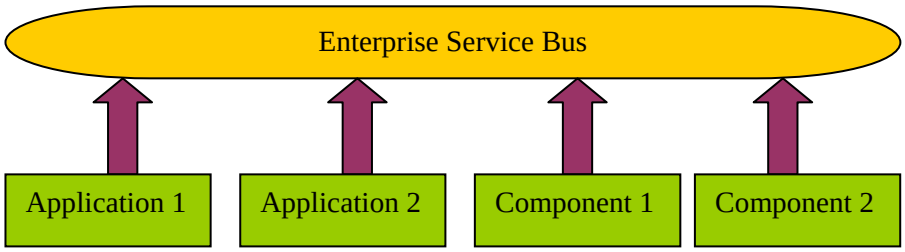


**Figure 12: Enterprise Service Bus**

### Rules engine

The rules engine is software able to infer logical consequences from a set of asserted facts or axioms. The inference rules are commonly specified by means of an ontology language and often a descriptive language. Many rules engines use “first order predicate” logic. Inference commonly can be performed in two ways: by forward chaining and backward chaining. Generally speaking, rules engine in the MPOWER platform is an observer that is gathering all information travelling through ESB. When some specific value or values match with some of the rules stored in the rules engine then a predefined action is carried out as a result.

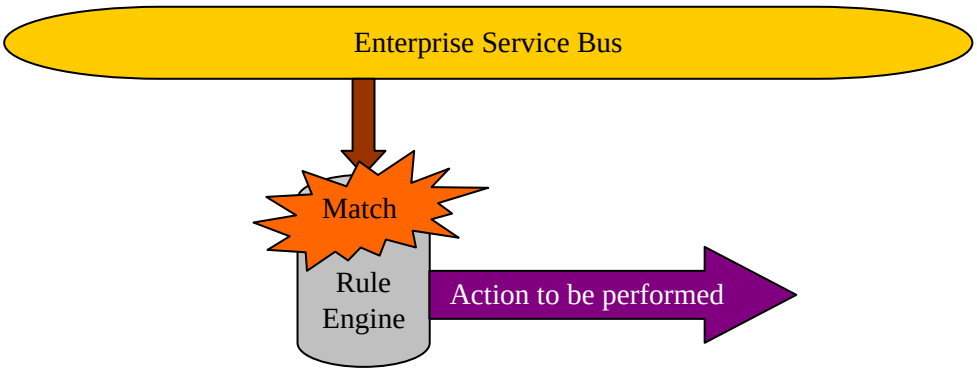


**Figure 13: Rule engine operation**

### Alarming

It is a service providing necessary logic of alarms treatment. It stores the necessary alarm's information and also passes a notification message to the applications which have to take appropriate actions as a response. These actions are described by a business process specific for each situation.

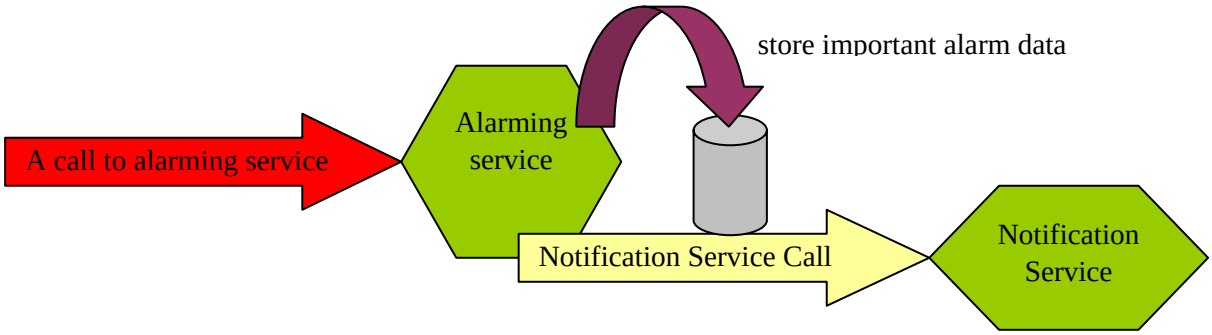


**Figure 14: Alarming service mechanism**

### Data bases

Data bases are needed to store necessary information within the MPOWER platform. Such information includes e.g. installed devices, user's data and preferences, etc.

### Frame Sensor Adapter

FSA (Frame Sensor Adapter) is proposed by MPOWER as a Sensor/Actuator Network abstract architecture model. It solves the problem of communication among services and different sensor protocols.

FSA architecture defines a framework to access sensors and actuators through several communication channels with the same interface. Regardless the information source and data format the information is managed by the FSA is unified. The goal of the FSA model is to hide complexity and details inherent to the sensor communication from the remaining system.

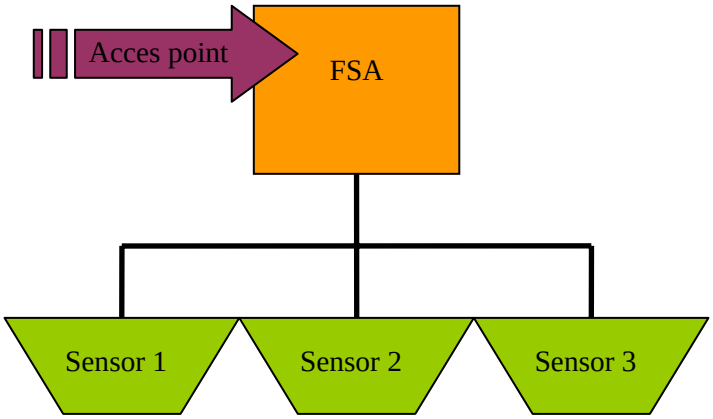


**Figure 15: FSA architecture, with unified access**

## 5.3 How to use a MPOWER service

### *5.3.1 Overview of the process*

The procedure of application development foreseen using MPOWER is a traditional process of user scenario evaluation - use case description - use case modelling – service modelling – service description – application development. This process is illustrated in the figure below:

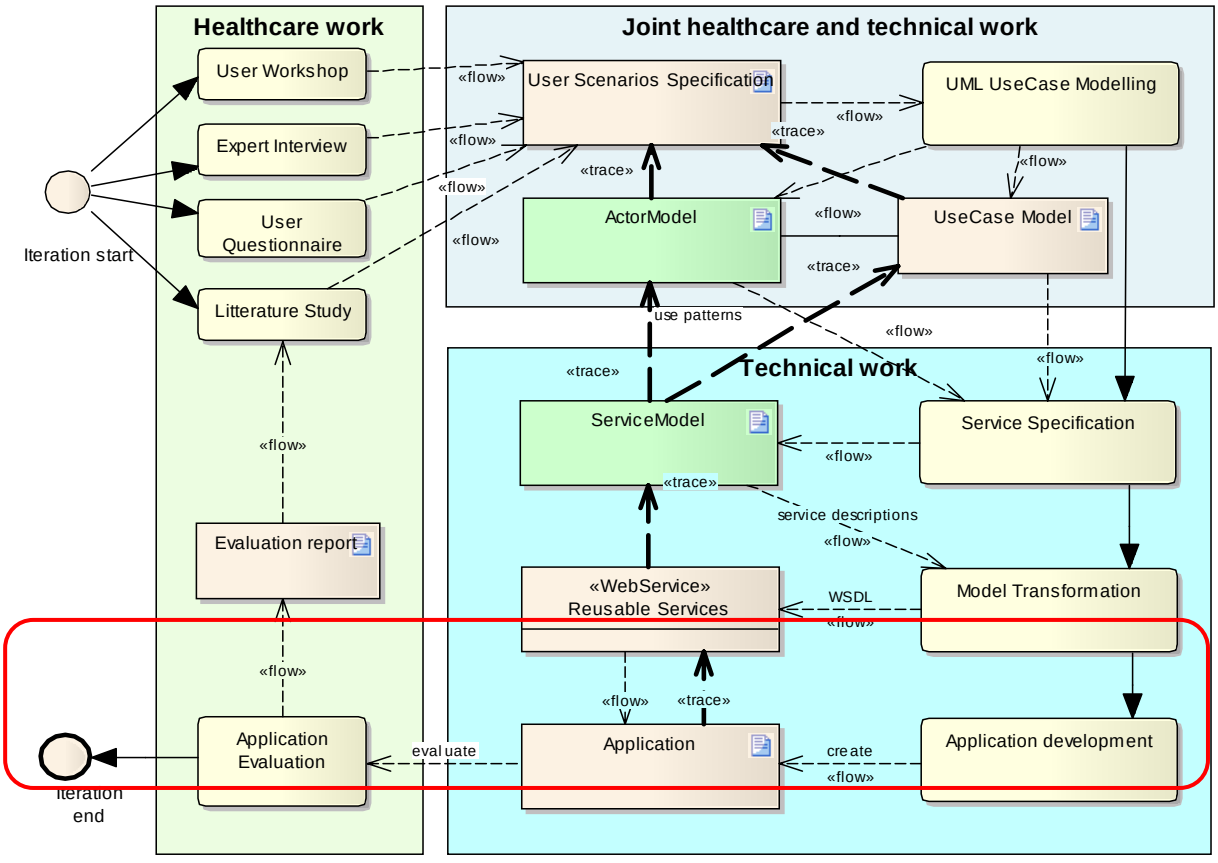


**Figure 16: Methodology of service and application development**

In this process the MPOWER middleware provides several services in the service model which are the basis to perform the application development. Therefore it is important to know all the services available and their description.

The figure below shows how the application can provide and use services. Every service has its specified input and output parameters. The “use” – relation describes the coherences between the applications. The signature of the services is based on the parameters which have explicit data types.

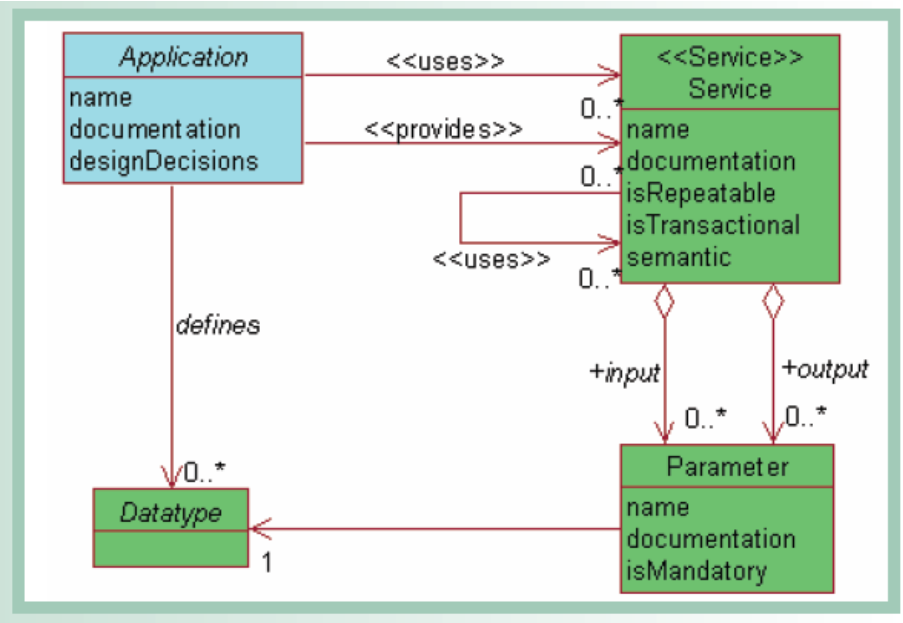


**Figure 17: Detail of an example UML profile**

## *5.3.2 Access MPOWER Information services*

Under the general information services set we can group the management services, communication services and security services, as well as information services that do not have to deal with HL7 messages and sensor services at the services layer.

The Services layer (from Figure 2: MPOWER Reference Architecture, section 3.3) includes the services the business chooses to expose. They can be *discovered* or be statically bound and then invoked, or possibly choreographed into a composite (business) service. The underlying service components provide service realization using the functionality provided by their interfaces which get exported out as service descriptions in this layer for use.

The Business Process Layer includes compositions and choreographies of services exposed by the Service Layer. The Services are bundled into a flow through orchestration or choreography, and thus act together as a single application. These applications support specific use cases and business processes, and are to a certain degree application specific.

The Proof-of-concept Applications (application layer) need to provide a user interface that uses the underlying (business) services. It is important to note that SOA decouples the user interface from the

components. Applications can call simple services at the services layer or choose to create more complex business processes.

The services at the services layer can be called in two ways. The first way is to use a service directly by creating a web service client that calls the service using the service's URL, the URL of a directory to which this web service's WSDL and any dependent files are published during deployment [11]. However one main part of the SOA paradigm besides the service provider and service consumer paradigm is to achieve loose coupling through the service broker and a service registry. So the other way to invoke a service is to use a service registry/broker system. The service provider creates a web service and possibly publishes its interface and access information to the service registry which is responsible for making the web service interface and implementation access information available to any potential service requestor. The Universal Description Discovery and Integration (UDDI) specification defines a way to publish and discover information about web services. The service requestor or web service client locates entries in the broker registry using a service discovery component (which uses various find operations) and then invokes the requested web service. The UDDI registry is explained in detail in the deliverable "D4.2 – the interoperability middleware design".

### *5.3.3 Access MPOWER Physical level services*

The MPOWER platform serves to manage information coming from many different sensors mounted in the place of its installation (smart house residence). This information is very important, because it constantly feeds the MPOWER knowledge base regarding: important healthcare variables ( such as heart rate, oxygen saturation, etc.) of the patient, environment state (temperature, doors and windows status, humidity, etc.), alarm situations (pressed panic button, location of a patient in case of getting lost, etc.) and more information to be taken into account. One problem of managing of such a number of sensors/devices and their returning values is that each sensor/device behaves in a different manner. By the sensor behaviour is here understood its usage through its communication protocol, data format used for exchanging messages, etc. That is why it is not possible to handle all of them the same way. This situation can bring a lot of problems at the time of adding new sensors. When a new sensor is added, if there is no other equals, developer must spend time learning the new sensor/device's behaviour which main consequence is the increment of implementation time.

Given these challenges, MPOWER provides support for these processes. We have created a solution that offers one single way of access to several kinds of sensors working under the MPOWER platform. The option adopted is to create a middleware, that we have denoted Framework Sensor Adapter (FSA), which is able to interoperate with several kinds of sensors and make them accessible in a standardised way. Developer only needs to have knowledge of how to communicate with middleware and not of how to communicate with all types of sensors/devices.

The main idea of the FSA is to offer a MPOWER service for each kind of sensor/device. MPOWER services are designed to talk to sensors/devices using the standardised protocol that the FSA defines. It is possible to offer the direct access to FSA, but it is more intuitive to use a service with a name related with sensor/device. It means that it is easier to remember that for consulting a door status there is a service called "DoorStatus" than one coming from FSA internal naming. For more details on the FSA, see the "D2.2" [7].

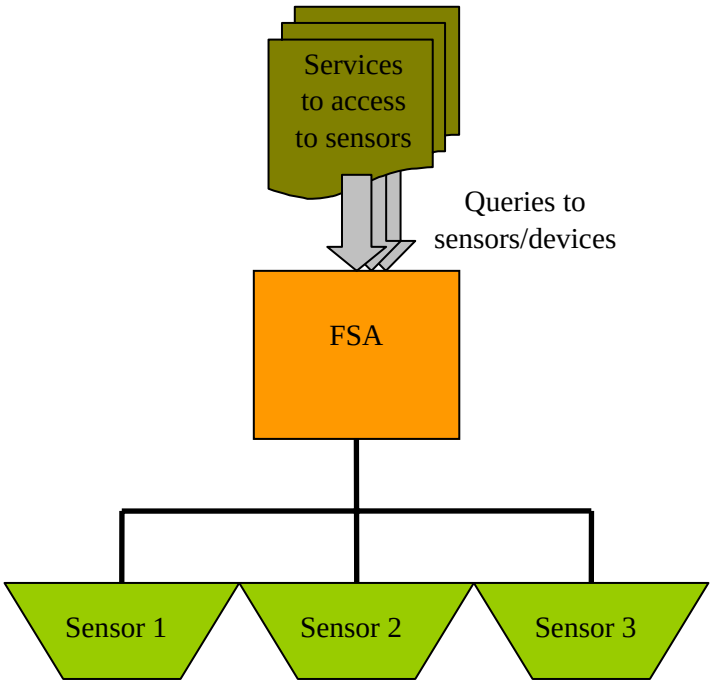


**Figure 18: MPOWER services for querying sensor**

MPOWER services are published as web services, so the sensors/devices returning values are accessible via SOAP requests. The platform offers a set of web services to access all different sensors/devices that can be found at the MPOWER platform deployment location. MPOWER services readily available within the platform are described below.

**DoorControl service**
Service designed for doors controlling. This service is able to ask for door status and to carry out actions against a door, such as open a door. It enables to developer to have knowledge of current situation of doors status (opened or closed), to get records of all movement of all doors and control door status.

**TemperatureControl service**
Service for temperature control in the “smart house”. Through this service one can get the information of temperature which comes from a particular temperature sensor and take an action to modify temperature. It enables to developer to have knowledge of current temperatures within different rooms, to get records of all temperatures of all rooms and control temperature in each room.

**CameraAccess service**
Service for getting a visual contact with a place of house via means of an IP camera. Thanks to IP cameras located in different places of the home, user gets current visual information of what is happening inside the home. It enables to developer to get image from a camera and show inside of a room.

### *5.3.4 Accessing HL7 services*

#### 5.3.4.1 The Reference Information Model

HL7 is used inside the middleware to handle all the medical information data of the person of care. HL7 provides a whole standardized communication and information model. This is the precondition to

provide this medical relevant data to outside legacy systems and provide a standardized interface and communication model. Information of HL7 is available on www.HL7.org.

The RIM can be found in reference [12].

For a given healthcare domain, an HL7 version 3 specification is based on the Reference Information Model (RIM), a common and underlying modelling framework, and includes artefacts like: Use Case Models, Information Models, Interaction Models, Message Models, and Implementable Message Specifications.

HL7's RIM is a static model of healthcare information representing the aspects of the healthcare domain undertaken so far by HL7 standards development activities. The HL7 version 3 standard development process defines the rules used to derive domain-specific information models from the RIM and to refine these models into HL7 message specifications, finally generating XML schema definitions (XSD) associated with a particular Message Type.

Various tools are provided to the standards developer by HL7 to support that process. For example, the RIM content itself is stored in a custom repository, and it's available to standards and application developers who want to access modelling information and/or create additional tools.

MPOWER uses the RIM model for implementing a whole communication and information model for medical data messaging. The overall process of defining information models that are relevant for implementing advanced homecare services is presented on the **Error! Reference source not found.**. It is worth mentioning that this process uses “top-down” approach which means that it starts from business requirements and process definitions and refines them in a stepwise fashion down to a software implementation. This process is compliant with SOA4HL7 methodology and with MPOWER modelling methodology that is used through the whole project lifecycle.

More information about the service modelling of HL7 services can be found in the deliverable “D3.1 – Medical and Social Middleware Design”.

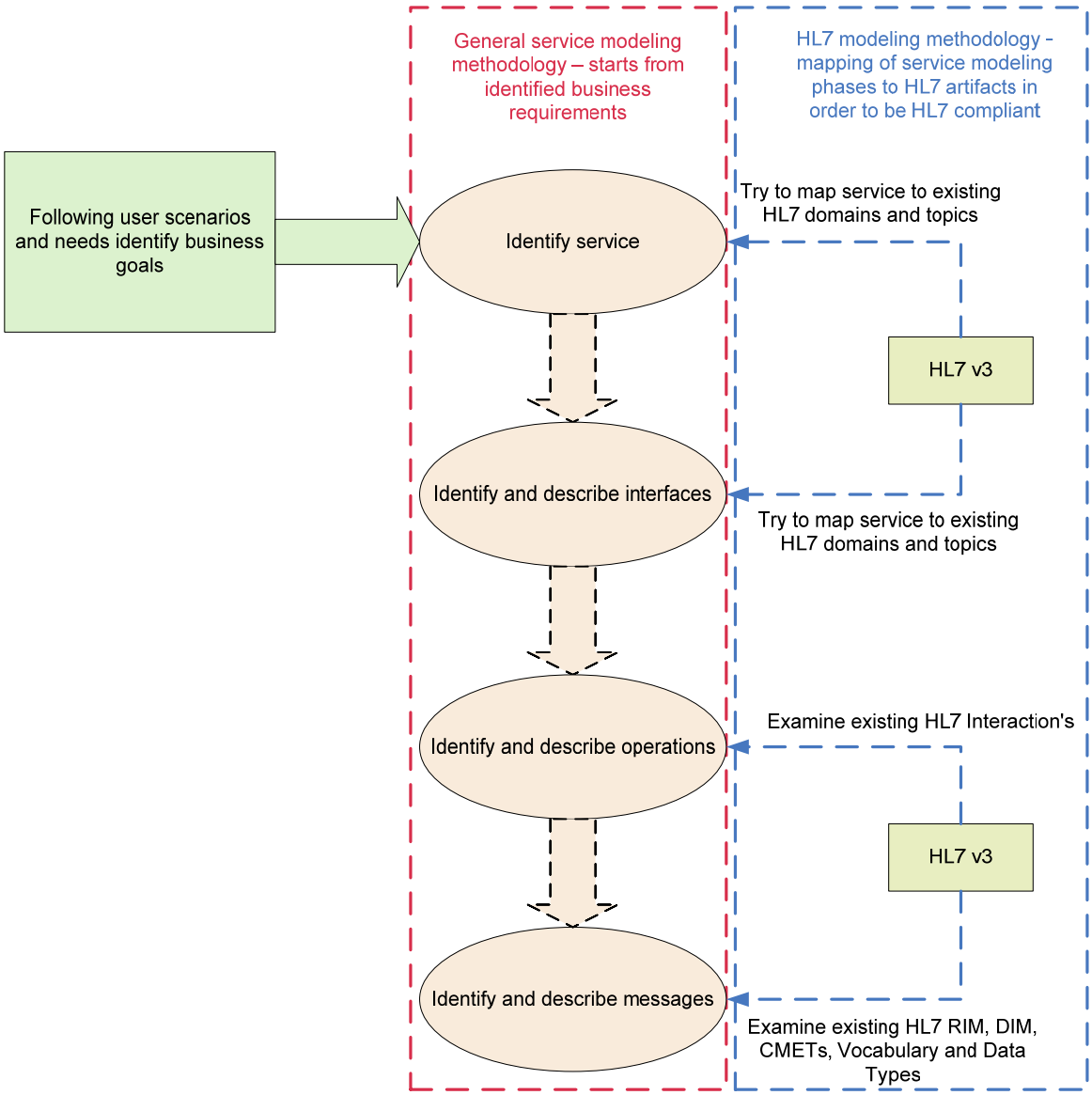


**Figure 19: Medical and social information modelling process**

### 5.3.4.2 The Message Structure

Interactions among HL7 applications happen through message exchanges. Thus, the standard provides a substantial level of functionality in provisioning envelopes supporting message exchange between applications. HL7 message envelopes are called wrappers, initially modelled by defining classes and relationships in the RIM. These specifications are then used to create the XML schema for the message wrappers, following a process outlined in HL7 Message Development Framework depicted on Figure 20 .

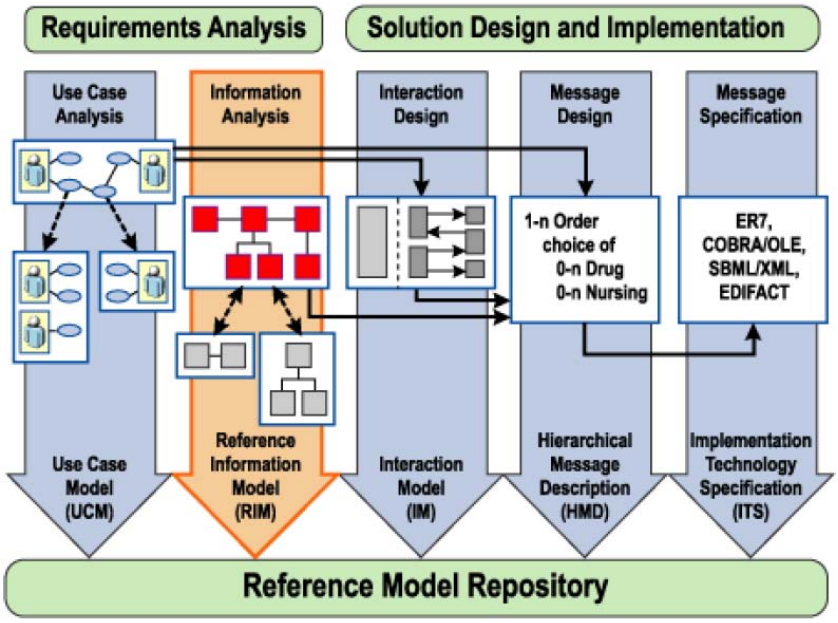


**Figure 20: HL7 Message Development Framework**

All HL7 messages are embedded into a Transmission Wrapper, whose goal is to support the Transmission (and Acknowledgement) of messages among applications. Important parts of the wrapper are elements like: Message Identifier, Message Creation Time, Interaction Identifier, Sender and Receiver Applications Identifiers, Accept Acknowledge Code, and Message Sequence Numbers (optional). It's important to clarify that HL7 messages should be thought of as being exchanged among Logical HL7 Applications; that is, specific software applications or components (like "Order Entry") that act on behalf of organizational or administrative entities (like "Westside Hospital Registration"). So, Sender and Receiver concepts should not be seen as part of a specification at the Transport level.

For example, the Accept Acknowledge Code specifies if an Application Level Acknowledgement should be returned to the application that sent a message, or if a more Transport-oriented receipt (called Accept Acknowledge) is required.

Sender applications use the Control Act Wrapper to communicate receiving information about the event that triggered the exchanged message. Unlike the Transmission Wrapper, the Control Act Wrapper's structure (schema) depends on the particular Interaction and Message Type. Not surprisingly, Control Act Wrappers are required for every message but Accept Acknowledge ones.

Transmission and Control Act Wrappers are used as envelopes for the Message Body (see Figure 21).

Together, the Control Act Wrapper and the HL7 Message Body constitute the complete semantics of HL7 Messages.

Inside the MPOWER middleware the HL7 messages wrap their content like explained above. Its clear that in the way of the design form the RIM only services designed in the process can exchange Hl7 messages. Nevertheless some services provide an interface where the information in the HL7 context can be retrieved in the HL7 structure. For more information see the deliverable D3.1.

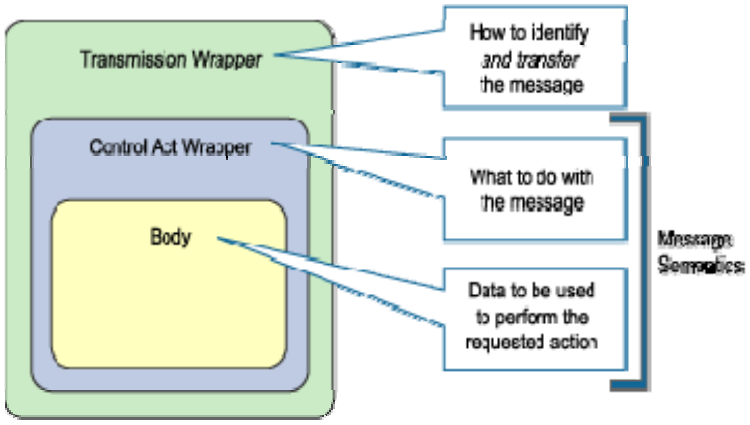


**Figure 21: HL7 Abstract Message Structure**

### 5.3.4.3 HL7 V3 Web Service Profile

The current HL7 V3 Web Service Profile specifies how to wrap full HL7 Composite Messages (including control act transmission wrappers) in SOAP and gives rules for certain aspects of the WSDL. Presented layers fulfil the purpose of providing a consistent way of transporting HL7 messages over SOAP and XML, and also define certain rules for defining consistent WSDL files (see Figure 22).

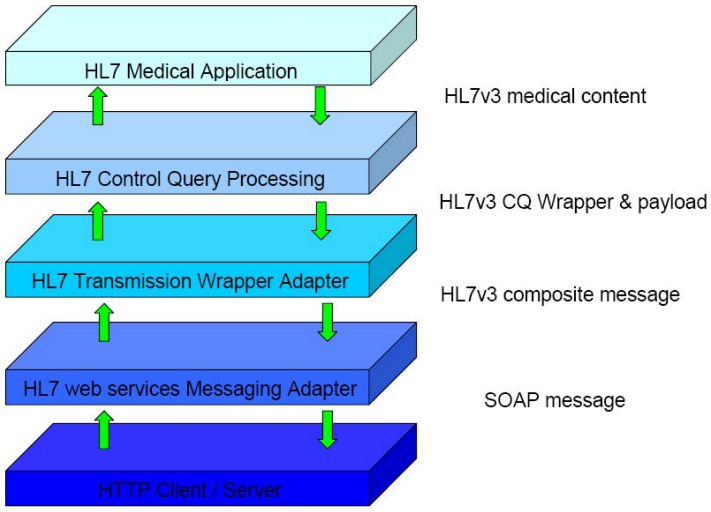


**Figure 22: HL7 V3 WS Profile Layers**

This approach has the advantage that from a HL7 perspective it uses the same construct as other "transports" and it makes transporting HL7 messages across multiple protocols simpler.

However, it does create any challenges:

- It is based on a messaging perspective rather than a SOA perspective and may limit the resulting benefits
- Web Services are treated as more of a simple transport mechanism based around a messaging paradigm.

HL7 messages within a SOA framework may be handled differently from other content types using a content specific infrastructure, resulting in inconsistencies, potentially less flexibility and slower and more costly development.

Inclusion of explicit Transmission layer adds unnecessary complexity and message bulk

Explicit dependencies and/or overlap between HL7 and WebService standards may create a continual synchronization challenge between the different standards.

#### 5.3.4.4 MPOWER HL7 services

MPOWER has a set of services that support the HL7 standard. For demonstration two information models have been used to develop middleware services.

Medical and social information models have been developed based on the HL7 RIM and in most of the cases constrain/extend/annotate already HL7v3 information models to support the needs of patient and professional caregivers. HL7 will be used as a basis to further develop models in the area of advanced homecare.

**Medication management service**

The requirement for medication management functionalities came from the interviews with patients suffering of cognitively disabilities and nurses that treat them. The interfaces and operations for the service are detected and selected from the HL7 services ballot [8]. Finally, the message contents are generated using HL7 tooling.

The Medication management service deals with the description of messages medicines for the purposes of messaging information about patient medication list. Operations of Medication management service range from basic functionalities for managing medication list to operations for fetching medical information from the database. The operations of the Medication management service are:

addDrugForPatient – adds new drug to a patient's medicine list,

deleteDrugForPatient – deletes drug from a patient's medicine list,

retrieveDrugList – retrieves drug list for requested patient, and

retrieveDrugInfo – retrieves detailed information about medication (description, package type, using guidelines etc.).

More documentation on this service can be found in the deliverable D3.1.

**Calendar management service**

The requirement for calendar management functionalities came through the interviews with families of elderly that needed a way to organize everyday tasks of their loved ones. The selection of appropriate HL7 services and operations, and generations of messages is done equivalently as with the Medication management service. The Calendar management services enable communication of events related to the scheduling of appointments for healthcare services and scheduling of different social activities. The scheduling system maintains a set of schedules related to certain person or resource, managing the process of arranging and booking appointments. The operations of Calendar management services are:

- bookActivity – writes new activity into a user's schedule,
- cancelActivity – deletes activity from a user's schedule,
- retrieveExistingActivities – retrieves the list of existing activities from a requested user's schedule, and
- retrieveActivityInfo – retrieves detailed information about activity.

More documentation on this service can be found at the D3.1 deliverable for a general HL7 service implementation process based on RIM models and the modelling example for the medical management service and the calendar management service.

## 5.4 Example development process using a scenario

### *5.4.1 Basic scenario*

Rosa lives alone in a small house in a village. She was diagnosed Alzheimer's disease 3 years ago and she is aware about the decline of her cognitive abilities. Some neighbours support Rosa with activities and she is more or less included in the social life. Every two weeks Rosa has to see the doctor to check her health condition. Due to the fact that medications she is using (number and sort of pills when they should be taken) are often changed, during the meeting the doctor gives Rosa a lot of additional information about what she should take care of (e.g. avoid some edibles, which side effect some of the pills have etc.).

In this basic scenario, the POCA system maintains the list of medications that are prescribed to Rosa. Rosa can at any time access the system and get specific information about medications that she is currently prescribed to. This additional information are entered by doctor and can contain various data related to the medication use. Furthermore, POCA maintains a scheduling system that maintains the schedule when Rosa needs to take the medication. The POCA system can initiate a reminder for Rosa to take the medication at appropriate time.

### *5.4.2 Upgraded scenario*

In the upgraded scenario, Rosa's condition has worsened, and Rosa sometimes forgets to take her medications even though she receives reminders from POCA system. Thus, POCA needs to be accommodated to this change.

### *5.4.3 Realization of basic scenario through business process*

The basic scenario can be implemented using business process that coordinates the Medication management and Calendar management services as presented in Figure 18.

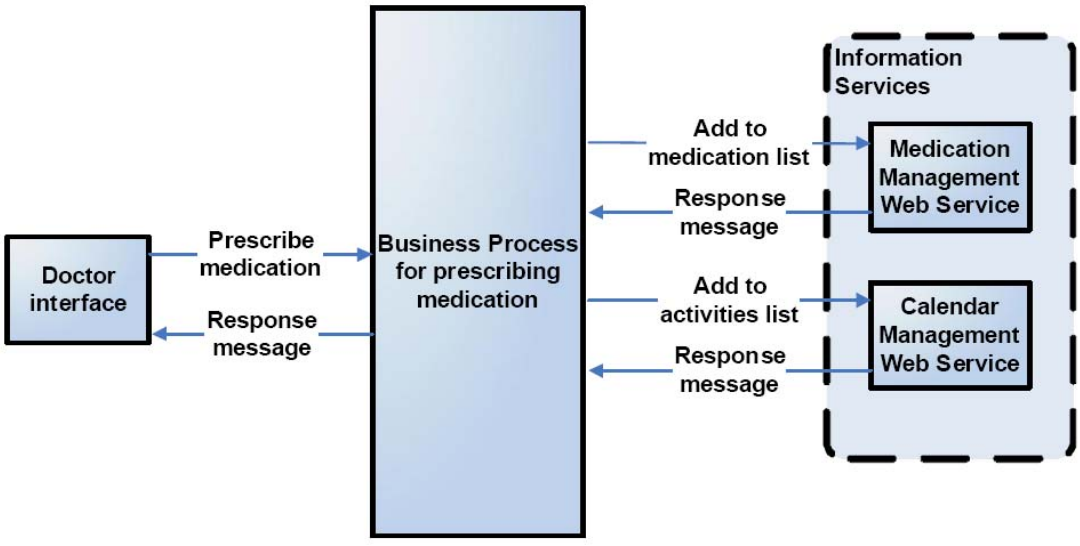


**Figure 23: Business process for prescribing medication to a patient**

The doctor uses Doctor interface to enter medications into Rosa's medication list. The Doctor interface uses business process to submit a medication to Rosa's list of medications. While submitting the medications the business process not only inserts the medication into Rosa's medication list using the Medication management service, but also makes a call to Calendar services where it schedules activities for Rosa. The scheduled activities contain reminders that will remind the Rosa when she needs to take the medication. Similarly, Rosa has user interfaces that she uses to access the systems services. The interfaces need not access the systems' services through business processes. For instance, interface for Rosa can access MPOWER services directly and display the list of medications prescribed to Rosa.

### *5.4.4 Adaption of scenario through business process*

After Rosa's condition has worsened the POCA logic needs to be modified to accommodate to additional functionality that is required from the system. For instance, the system may be adopted in a way that Rosa is assigned a caregiver that is willing to help her to take the medications. The caregiver need to be updated of any modifications to Rosa's list of medications or schedules.

The upgraded business process coordinates three services. New service that business process uses is Message sending service from Interoperability services package. The Message sending service cooperates with telecom provider and enables sending SMS messages. When a new medication is prescribed through the new business process, the process not only adds medication to the list of medications and schedules a reminder, but also sends an SMS to the caregiver notifying him of the modifications to Rosa's medication plan.

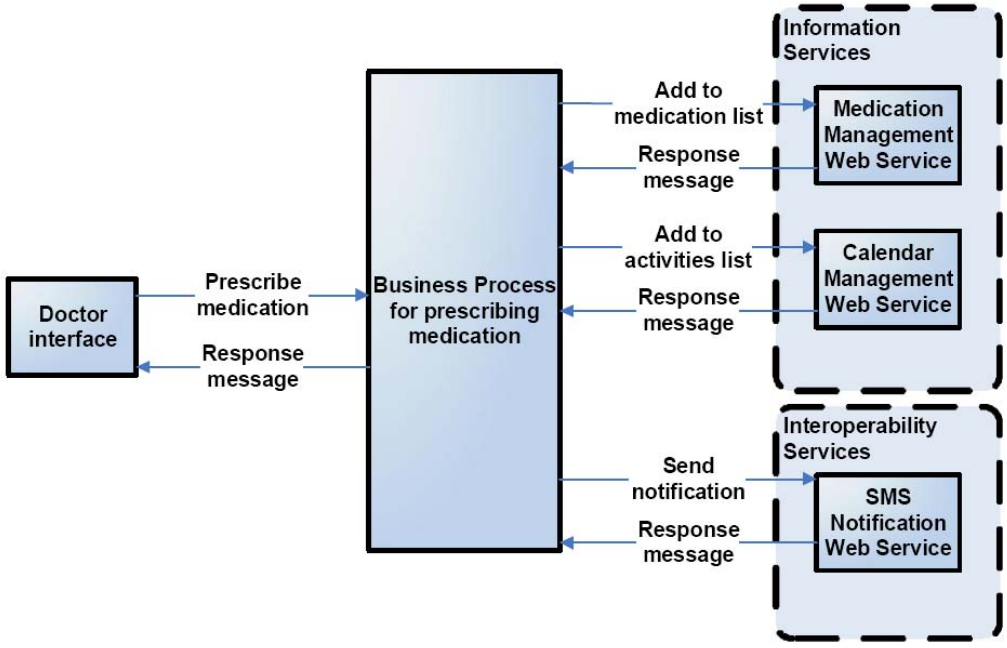


**Figure 24: Adapted business process for prescribing medication to a patient**

## 5.5 Step by Step guide for creating a small application

In this chapter we will demonstrate how developers can create an application and calling a web service from the MPOWER platform. In the specific example we will use Netbeans, the application will be a web application and the web service that will be invoked it will be the authentication service.

The prerequisite for this task is to know the wsdl of the web service. In our case the wsdl of the authentication service is:

http://mpower2.arcsmed.at:8080/sec.access_control/AuthenticationWServiceService?WSDL.

First of all, we should create our application, if it's not already exists. We open our IDE and select the create a New Project menu. In Netbeans it can be located on the upper left side of the IDE (Figure 25)

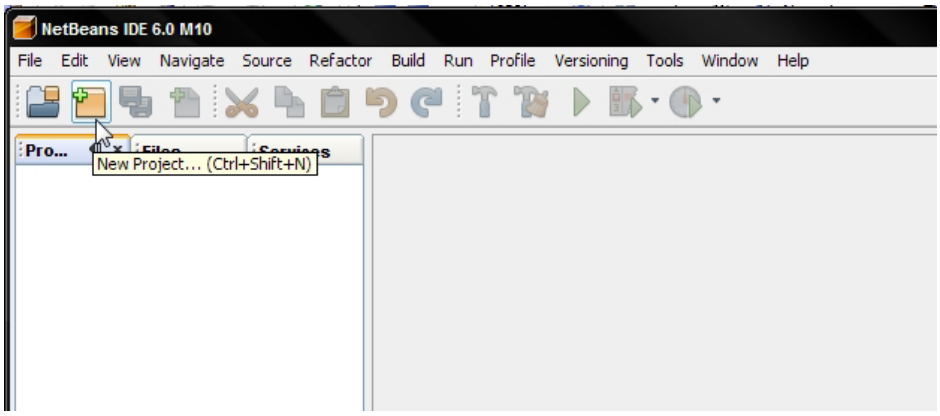


**Figure 25: Create a new project**

Then we select the type of the application. In our case we select the web application (Figure 26). Then we have a window to insert the application details (Figure 27). Mandatory fields are the Project name. All other fields are automatically inserted, but we have the option to alter them. By pressing the Finish button we have the IDE main screen and the new project opened and waiting for our code (Figure 28).

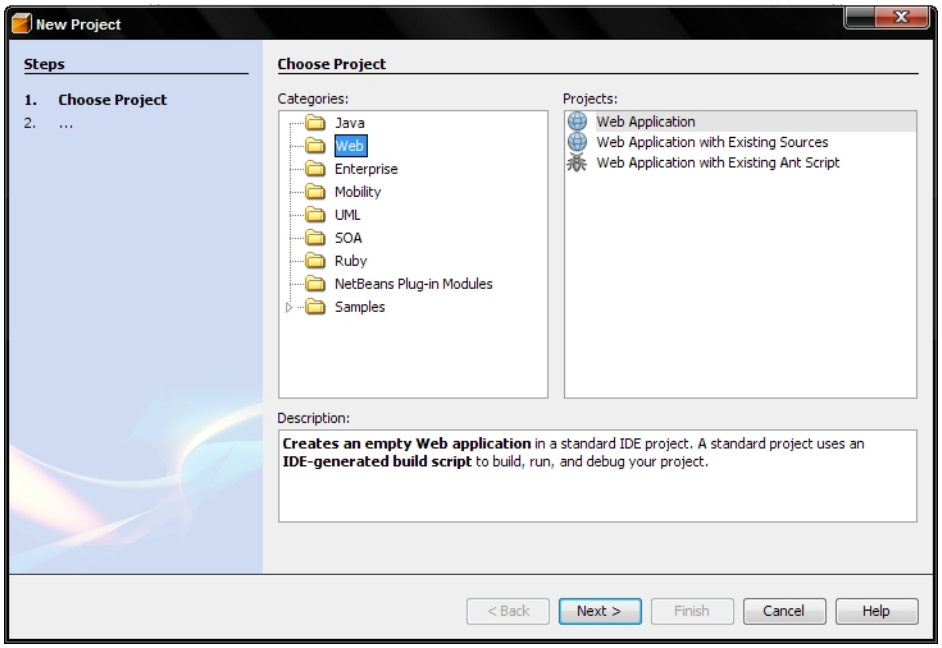


**Figure 26: Select Application type**

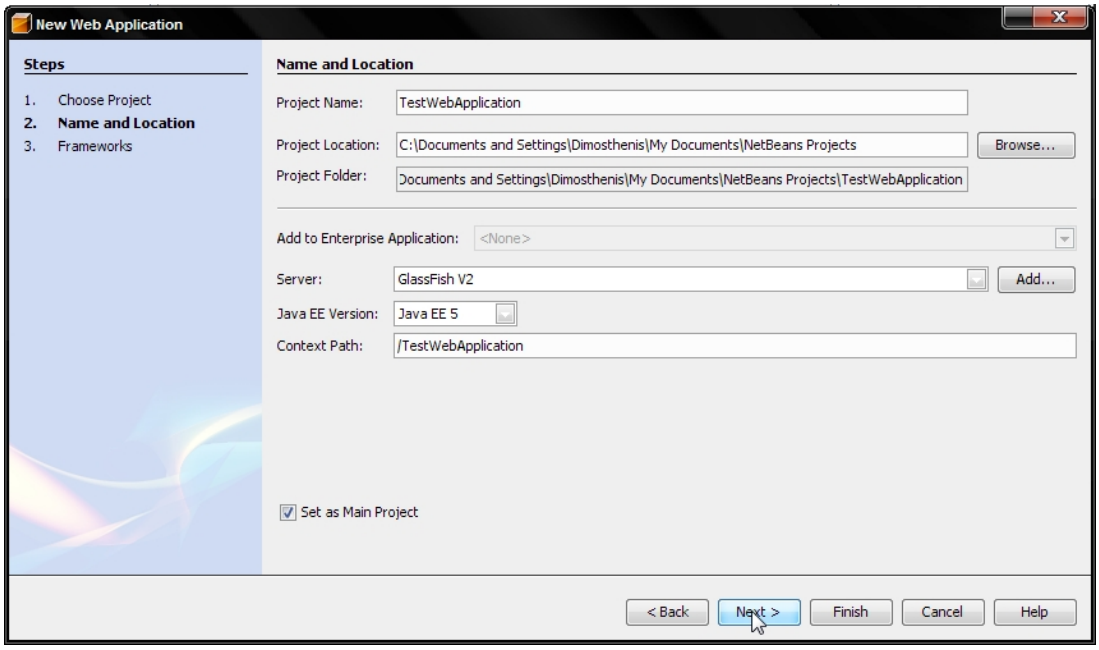


**Figure 27: Complete the Application details**

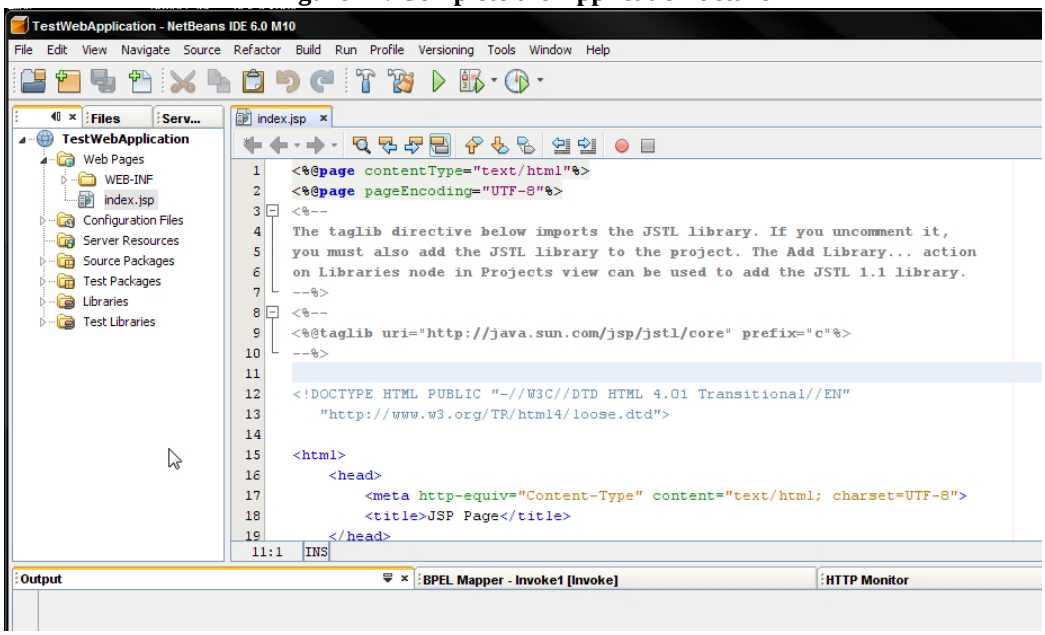


**Figure 28: Ready to begin the Application development**

The developer now creates the pages and inserts the code that desires. After few lines of code, we assume that we need to use a web service from the MPOWER platform. In order to do so, the developer has to create a web service client (Figure 29). When selecting the option to create a web service client, a window appears for the user to enter the web service details (Figure 30).

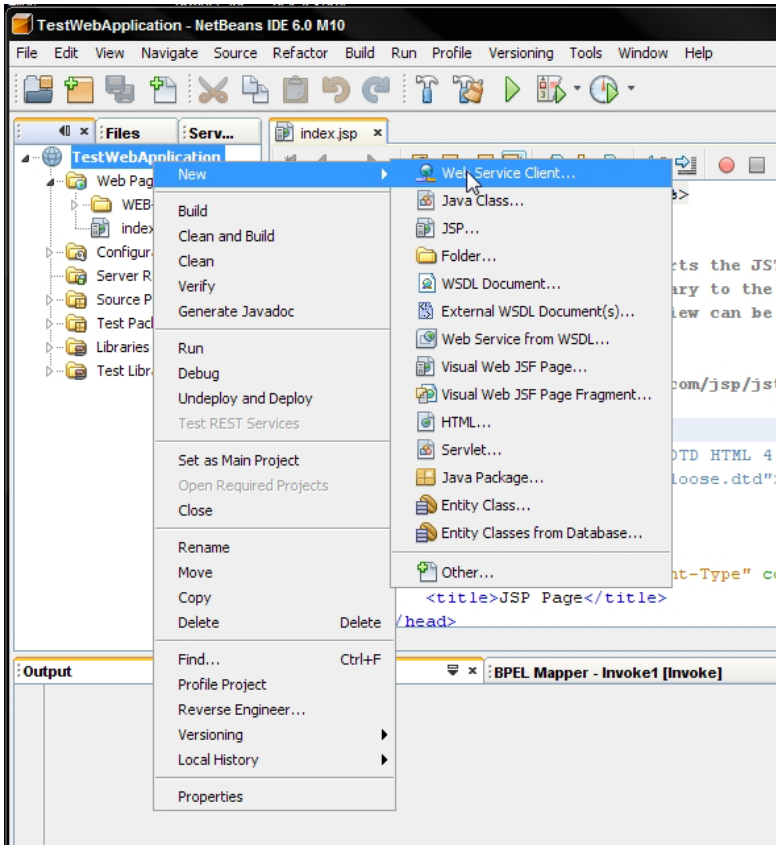


**Figure 29: Create a web service client**

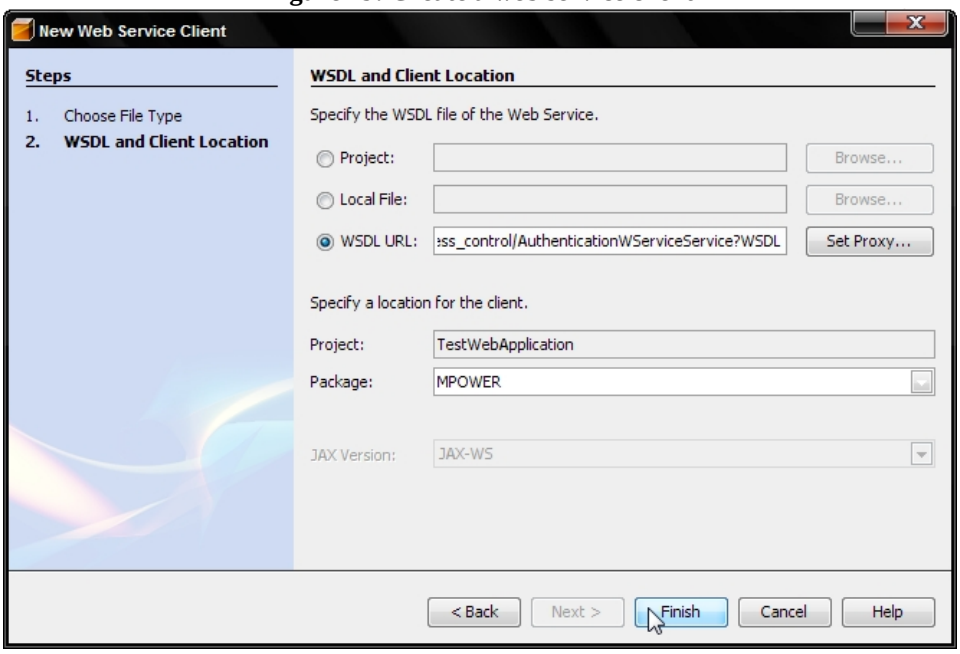


**Figure 30: Inserting details for the web service**

After the insertion of the web service client, the developer has to call the web service in order to use the MPOWER platform. We can easily do this by expanding the service and locating the desired procedure of the service and just drag and drop into the code into the right place (Figure 31). Then a ready code is inserted into our code (Figure 32). The developer is now ready to alter the code as he sees fit in order to succeed his goals.

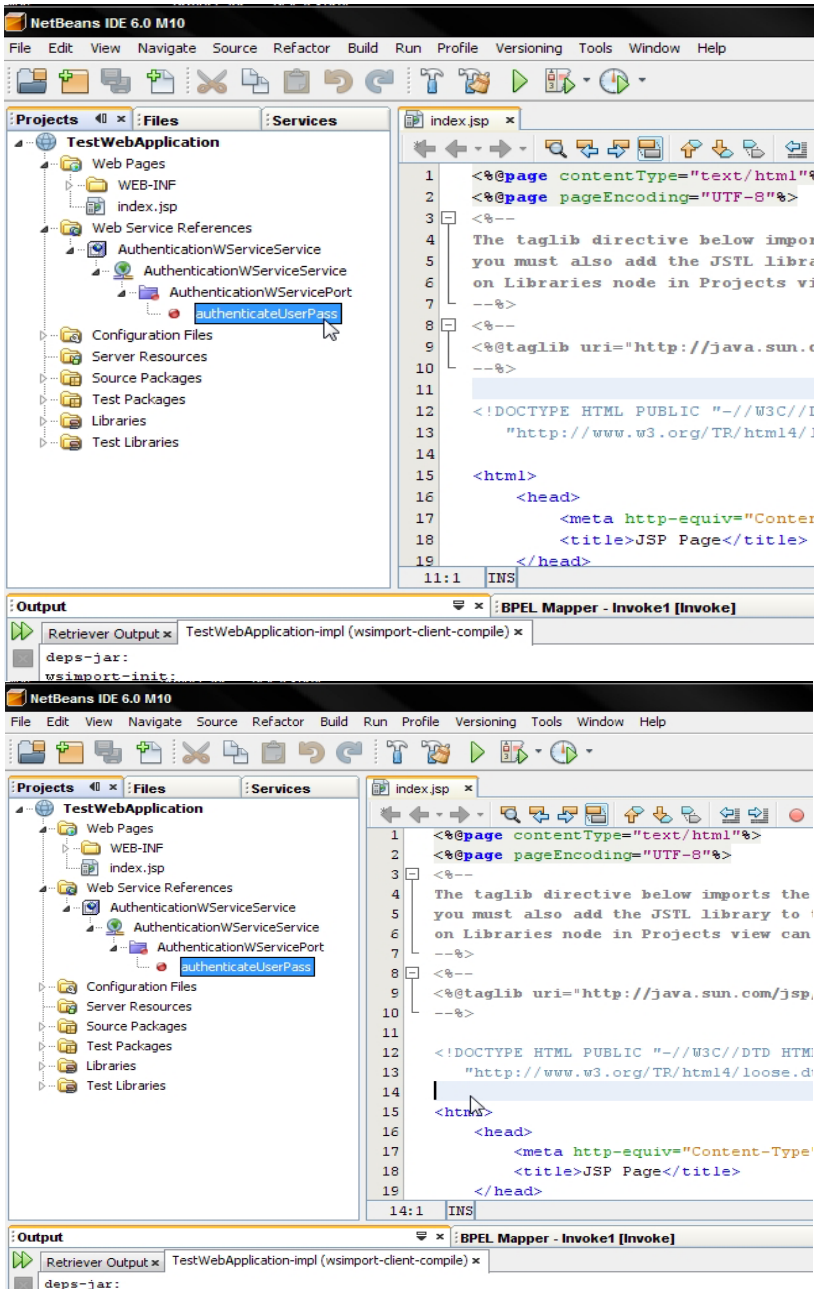


**Figure 31: Inserting the service**

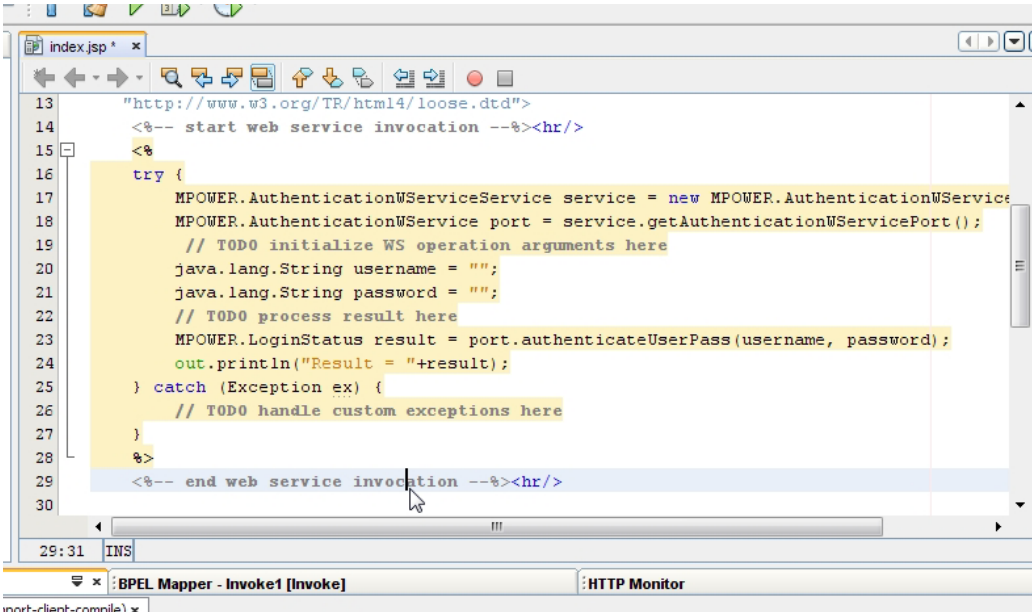


**Figure 32: Auto code from the IDE**

## 5.6 Creating and using a business component in BPEL

Business processes are modelled with the Netbeans environment using SOA projects category. The detailed tutorial on how to create a simple business process in Business Process Execution Language (BPEL) can be found here: http://www.netbeans.org/kb/61/soa/bpel-guide.html

# 6 Service Developer's guide to MPOWER

This chapter describes how a developer can create services which fulfil the requirements for extending or being interoperable with the MPOWER platform. The chapter describes the MPOWER service development methodology, and presents the tool-chain recommended for developing MPOWER services. The last sub-chapter presents step-for-step instructions for the design and implementation of an example service.

## 6.1 How to create a service

### *6.1.1 What is an MPOWER service?*

From the developer's viewpoint an MPOWER service is a standard web service based on SOAP messages and WSDL descriptions, and with some additional rules to follow:

- Services must follow the WS-I Basic Profile
- SOAP binding must apply the document style (not RPC)
- Services must use the MPOWER security mechanisms
- All services operations must use the MPOWER status return codes

When developing services in the MPOWER context, we can separate between two scenarios which differ in e.g. rationale, requirements, and usage. These scenarios are described below. Note that the general rules for developing MPOWER services described above apply to both scenarios. Also, the methodology for developing MPOWER services described in the further sections of this chapter is applicable in both scenarios.

Developing services which will become part of the MPOWER platform:

- **Rationale**: To become part of the MPOWER platform, the service should be generally reusable in different applications of the domain. Thus, care should be taken when designing the service to consider a set of different usage scenarios, avoiding to tailored the service for use only from a single application.
- **Deployment**: A service which will become part of the MPOWER platform should be deployable together with the rest of the MPOWER platform, avoiding the need of additional application servers for the platform.
- **Shared database**: The services which are part of the MPOWER platform share a single database, and thus new services which will become part of the platform should be developed to use this shared database in order to avoid the need of multiple databases for the platform.
- **Technology**: To be deployable with the MPOWER platform, the service should be developed in Java so that it can to use same application server as the rest of the middleware.
- **Namespaces**: MPOWER defines a namespace scheme to apply for the services which are part of the platform.
- **Logging**: Services which are part of the MPOWER platform should follow the standard MPOWER approach to logging.

Developing more independent services based on, or interoperable with MPOWER

- **Rationale**: Not all services should become part of the MPOWER platform. When developing services that have limited reusability, e.g. because it is more or less application specific, in order to keep the platform focused and manageable in size the service should not become part of the platform. Another reason for not adding a service to the platform (even though it is

reusable), could be that the service is developed as part of a separate product with different licensing terms than the MPOWER platform.

- **Used together with MPOWER services**: The independent services will typically be used in combination with services in the MPOWER platform to create applications extending the scope of the services provided directly by the MPOWER platform.
- **Deployment**: For independent services, it is not a requirement that the service is deployable together with the MPOWER platform. In this case, the service could be deployed independently of MPOWER using another node, another application server, and another database. Also for independent services, it could still be a benefit to have the service deployable with the platform, in which case the requirements for the first scenario should be fulfilled.
- **Technology**: Independent services can be developed in any technology for Web-service development (e.g. .NET), but note that the MPOWER tool support is only provided for the MPOWER-recommended technologies (Java-based).

### *6.1.2 Overview of the MPOWER service development methodology*

The previous subsection described what is required of a service in order to fulfil the requirements for being an MPOWER service. In this sub-section we provide and overview of the methodology for developing MPOWER services. The methodology described in this and the following sub-sections is independent of specific tools, but assumes the use of UML as the modelling tool. The model examples provided in the chapter was made using the MPOWER UML profile in the Enterprise Architect tool, but should be possible to reproduce using other tools.

Figure 33 presents an overview of the process that was followed when developing the services which are predefined in the MPOWER middleware. When developing new MPOWER services, we recommend that you use this process as a starting point, but also that you tailor it to your own needs. For instance, you may decide to use a user workshop as the only input to specifying user scenarios for your service (e.g. because of resource and time constraints). Below the process is described in more detail.

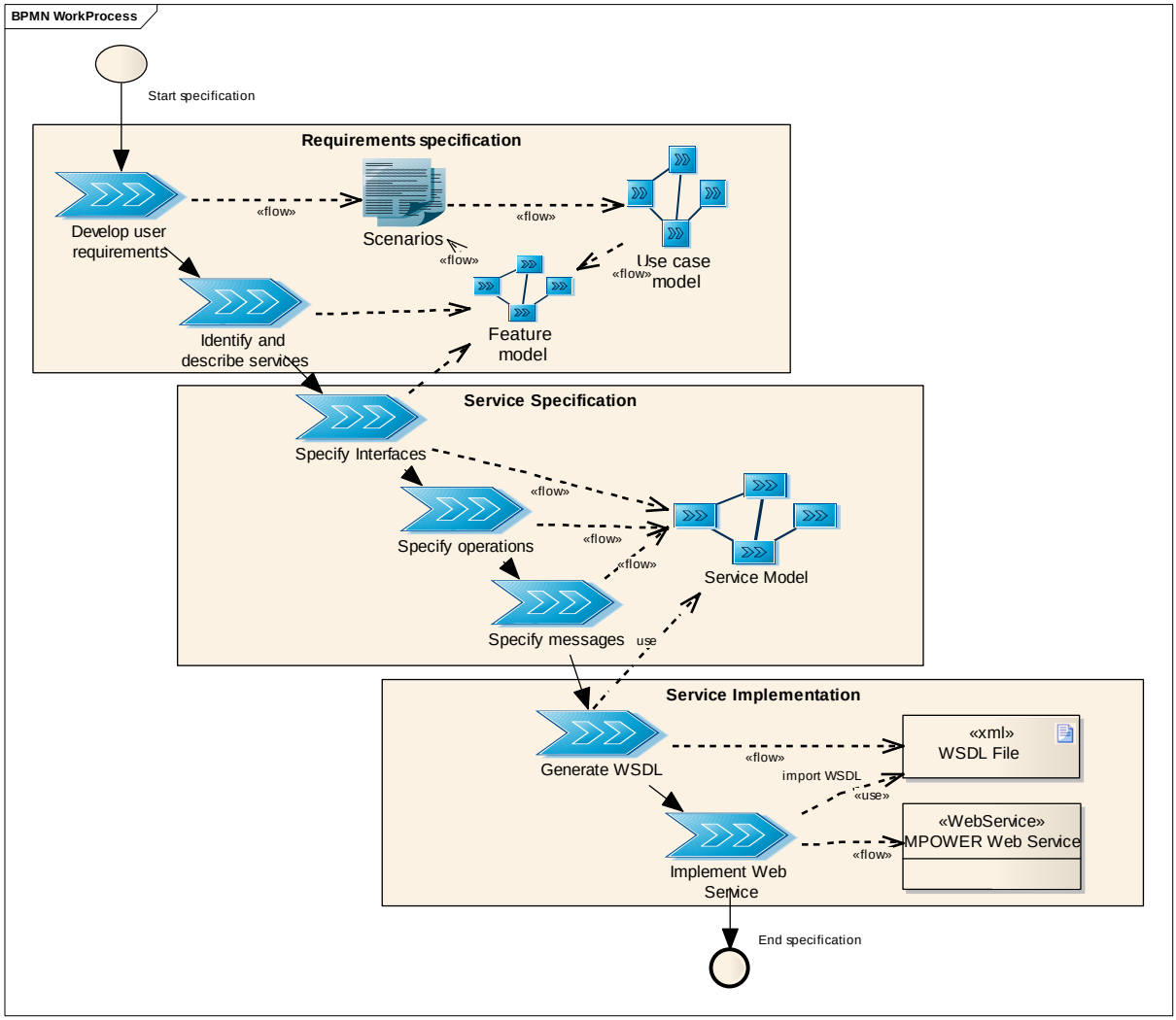


**Figure 33: Service specification process**

In order to initiate work to create new services we need new requirements from the end users. These requirements are developed using scenarios, use cases and features. A scenario is a story that first describes the problem and then describes possible solutions. Therefore these scenarios represent the basis for the development of the use case and feature models. User workshops, expert interviews, user questionnaires and literature studies may be applied to come up with a good set of scenario specifications. Figure 34 shows the requirements development approach applied in the MPOWER project.

The user scenarios will be modelled in UML use cases, which map to specific scenarios of use and help capture the functional requirements of the system (or *features* as we denote it in MPOWER). Use cases describe the typical interactions between the users of a system and the system itself, providing a narrative of how a system is used. If new actors are involved we will need to update the actor model as well. So from the user scenarios and the use cases we can extract the features (non functional and functional). The functional features are specific requirements for services to be implemented but are not detailed specifications. The features can be grouped and prioritized.

Using the user requirements (mainly the feature model) as input, service candidates are identified and described. After we decide the services that will be implemented we need to specify the service interface along with operations and messages for each of the services in a UML class model (service model).

Based on the service interfaces, the developer can now create WSDL for the services. With the proper tool support (e.g. Enterprise Architect), this can be done (semi)-automatically by applying

transformations to the UML service model. The WSDL definition of the service is then used as basis for creating the implementation of the web service, and the development environment (e.g. Netbeans) will typically generate proxy classes and skeletal implementations of the service interface in which the developer will fill in their code.

When we finish with the implementation and build the service, the development environment typically produces a deployable file (e.g. WAR file for Java) that contains all the class files. This file can then be deployed on an application server to make the service available for use in applications and other services.

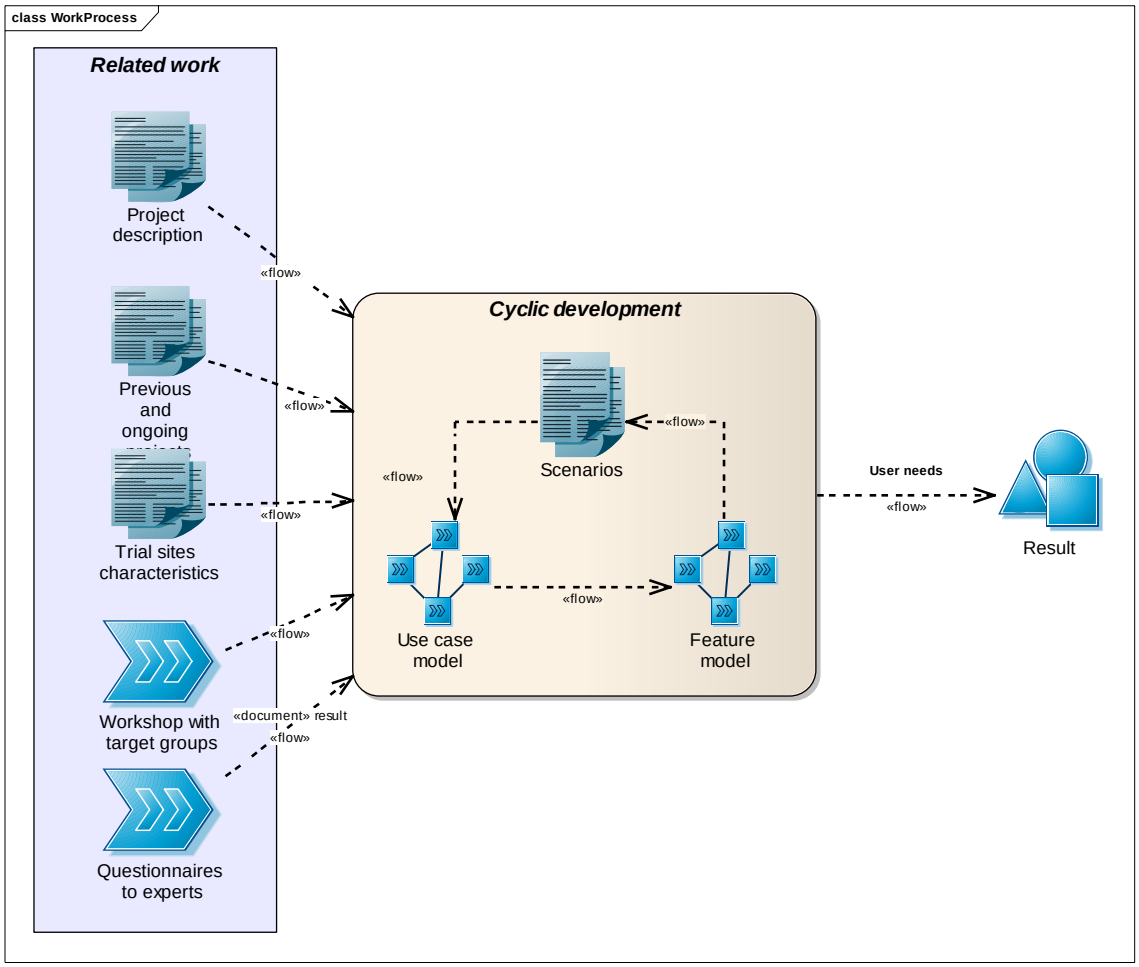


**Figure 34: The MPOWER User requirements development approach**

### *6.1.3 MDA approach*

To address the information system interoperability problems, new techniques and methodologies have been introduced in the Software Engineering community. One of these is the Model Driven Development, or more precisely Model Driven Software Development (MDSD). The goals of MDSD described in [9] can be summarized as follows:

- Increase development speed and software quality through automation
- Higher level of reusability as the architectures, modelling languages and transformations are generic for the domain (abstract)
- Improved manageability of complexity through abstraction
- MDSD is based on the Object Management Group's Model Driven Architecture ® (MDA). OMG's focus in on interoperability, portability and reusability through architectural separation of concerns

The development of the MPOWER middleware platform follows the model-driven approach as defined by the Model Driven Architecture from OMG [10]. MDA specifies three views of model development:

- Computation Independent Model (CIM): the domain (business) activities are modelled independent of the information system to be developed
- Platform Independent Model (PIM): the information system behaviour and structure is modelled, based on the CIM, independent on the technological platform and tools that will realise it
- Platform Specific Models (PSM): the system is modelled in UML with technology-details about the realization. From PSM the major parts of the system code can be generated.

In the following, the method for MPOWER middleware platform development method is presented in terms of these three views.

### *6.1.4 Capturing domain knowledge: User needs – scenarios, use cases and features*

As described in the overview of the methodology, the recommended approach for developing services based on MPOWER starts with capturing the user needs and business aspects (the CIM) resulting in a set of scenarios. A scenario is a hypothetical story, used to help a person think through a complex problem or system. The scenarios are written in natural language, and describe different situations in which the user will interact with the system. It can be useful to use two different kinds of scenarios: first a problem scenario which describe the current practice, and then an activity scenario where an improvement to the current practice is suggested by introducing use of a new system or service.

Based on the scenarios, a set of UML use cases are described next. In this process activities and actors involved in the scenarios are identified, and these constitute the content of the UML use case. Each scenario can result in one or more use cases, but in some cases a use case may also be based on more than one scenario. Figure 35 shows the “Stakeholder management” use case and how it is related to actors and other use cases.

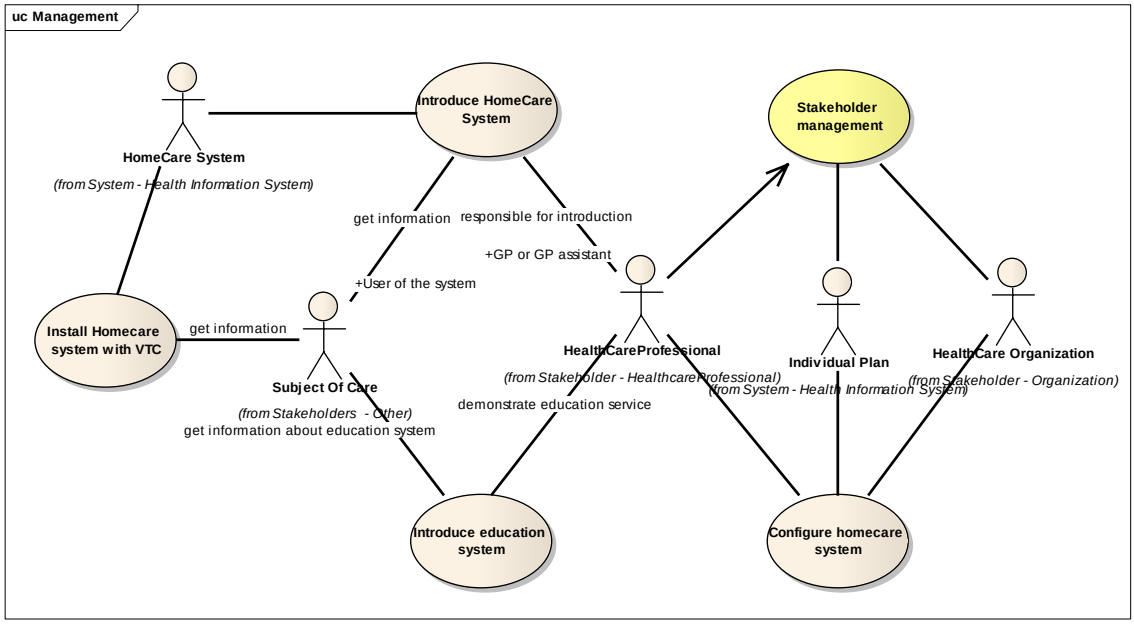


**Figure 35: Use case example from MPOWER Management scenarios**

It is strongly recommended to document the relations for how use cases are derived from scenarios. This allows traceability both for determining the origins of the use cases, and for checking how the scenarios are followed up in the use cases. Figure 36 depicts how the traceability to the scenarios from which the use case is derived can be described using the Enterprise Architect tool.

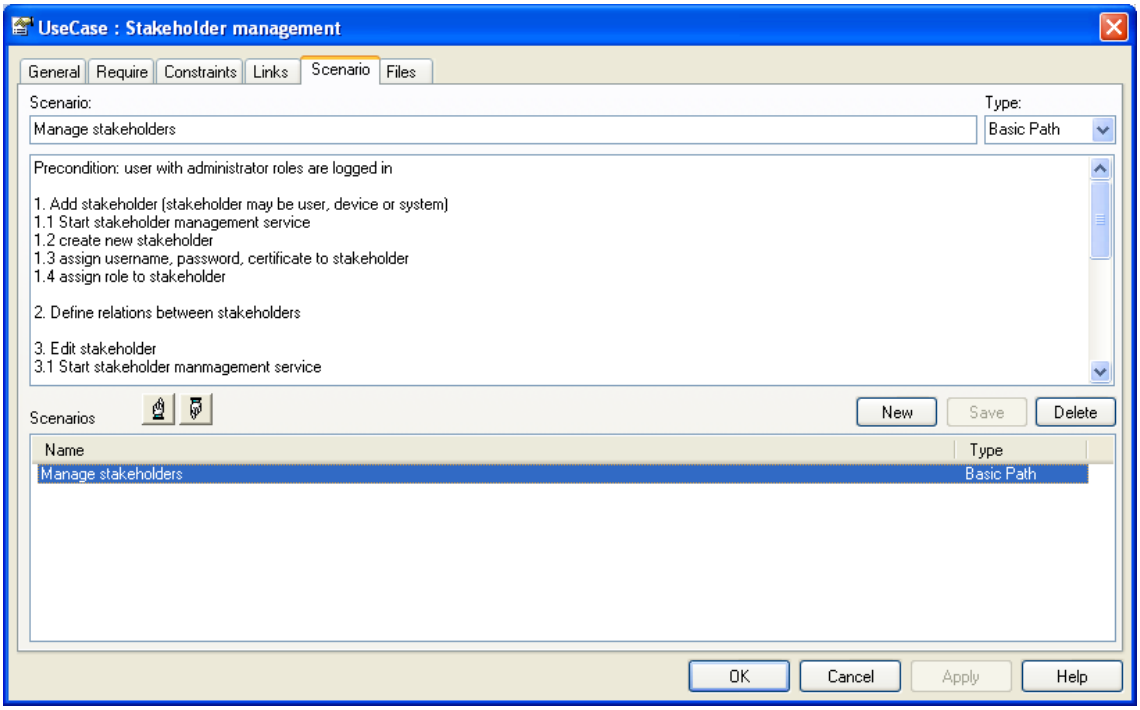


**Figure 36: Traceability from use case back to scenarios**

From the use cases and scenarios features are described and grouped into categories. Each feature should be directly related to one or more use cases as shown in Figure 37. In this example, the stakeholder management use case is related to four features.

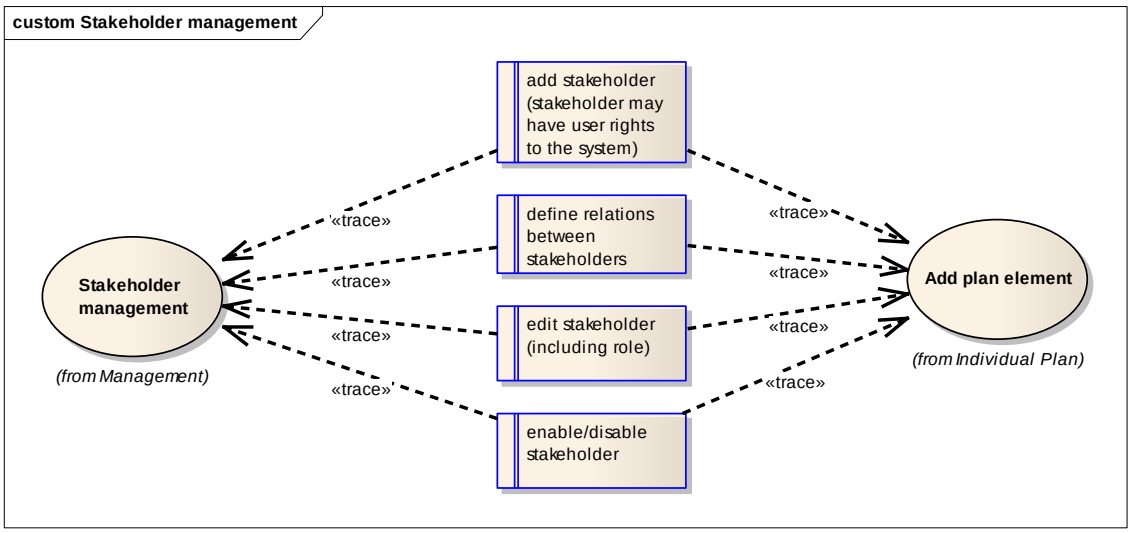


**Figure 37: Features and their relations to use cases**

Further description of the how scenarios, use cases and features were applied in the development of the predefined services of the MPOWER middleware can be found in the deliverable MPOWER D7.1, “Scenarios and needs”. Also, in the annexes to this deliverable examples of scenarios and use cases can be found.

### *6.1.5 Service Specification – the Platform Independent Models*

The features and use cases are used as input for defining the services. As an initial approach, related features can be grouped to form a service. The relation between features and services should be

documented in a UML diagram similar to Figure 38. This diagram shows will show the rationale behind the service, and provide traceability back to the features.

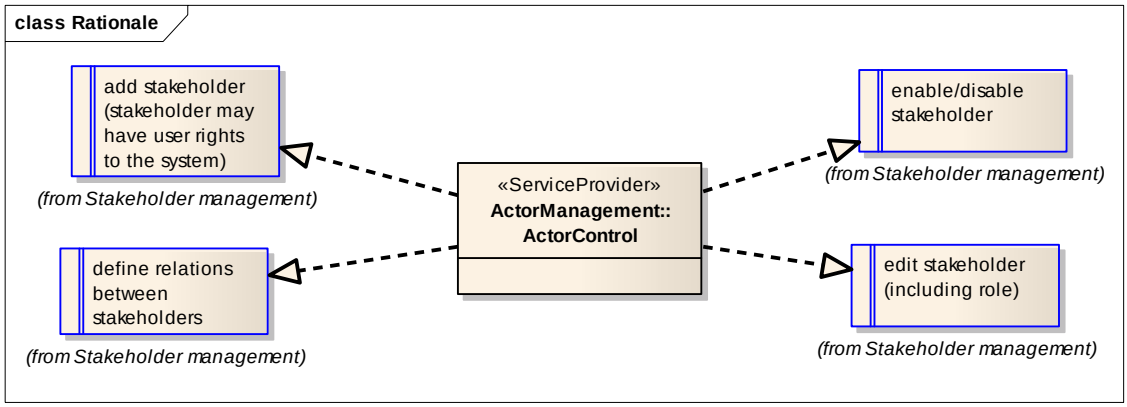


**Figure 38: Service rationale**

For the further development of the service specification, we recommend to follow the process described in Figure 39. As described above, grouping the features will be the main approach to identifying services.

Service definitions can include one or more interfaces. In general, different interaction styles can be split into different interfaces e.g. Query (read-only) vs. Manage vs. Notification (subscription based). In case there is only one interface it can be called the same as a service. The features will also give some input to the process of defining the interfaces and their operations. Experience from the development of the MPOWER services indicate that the tools support works best if there is only one interface for each service.

The service definitions are described using UML class diagrams. As shown in Figure 40, the service is defined by a UML class representing the service provider, a UML interface describing each interface and its operations, and with the interface associated with a port of the service provider.

The right-hand part of Figure 39 describes additional activities in the service definition process which should be used when defining HL7-based services. Further details on this process and HL7-specific issues can be found in MPOWER D3.1: Medical and Social Middleware Design.

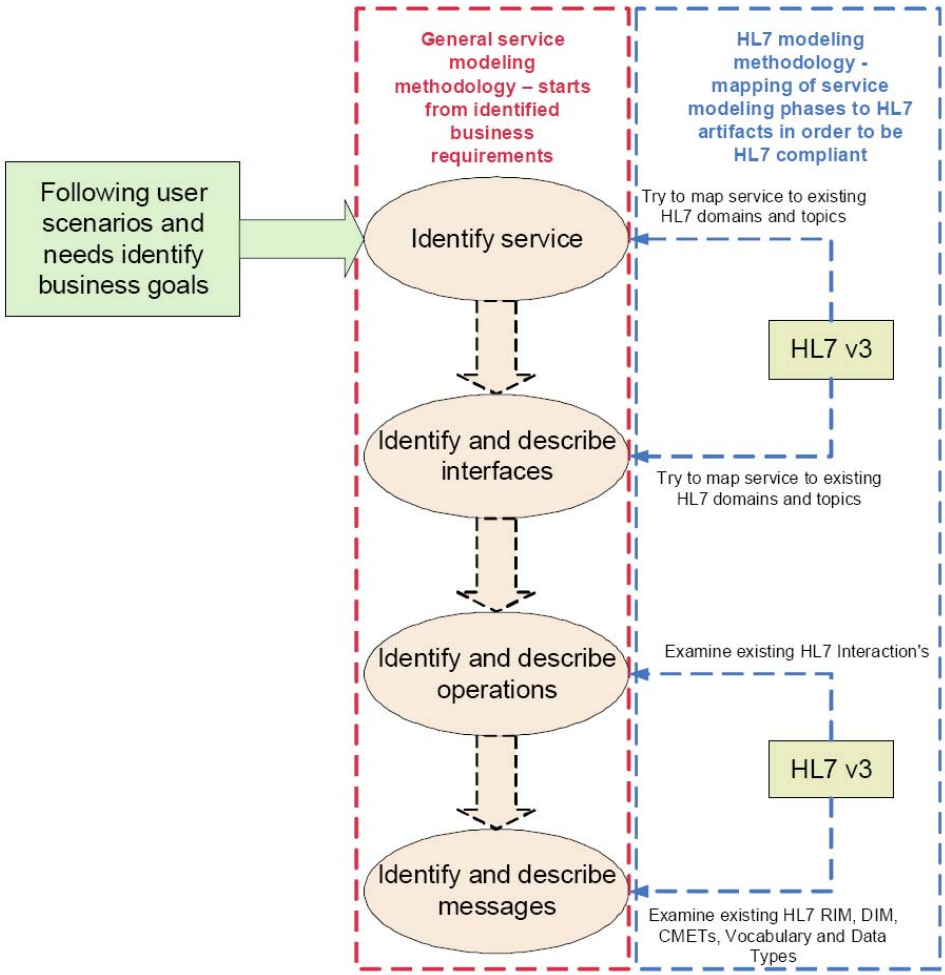


**Figure 39: Medical and social information modelling process**

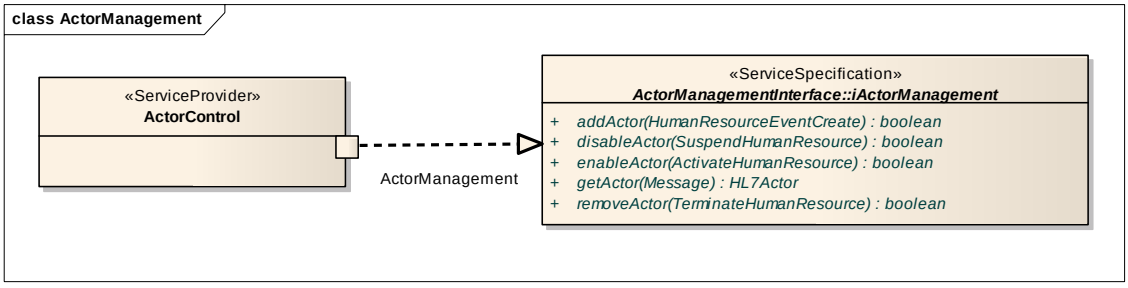


**Figure 40: The Service Model with UML Stereotypes**

## 6.2 Using the service developer toolchain

This chapter describes how you can use the MPOWER developer toolchain to develop MPOWER services. The first sub-section gives an overview of the MPOWER toolchain. It is followed by sub-sections describing details specific to the development of information related services, physical level services, and services using HL7.

### *6.2.1 MPOWER toolchain overview*

Using the MPOWER toolchain is the recommended way of developing MPOWER services, as it was constructed to support the development methodology and increase the productivity of the developer. The current implementation of the toolchain is built around the following tools:

- Enterprise Architect for UML modelling and transformations
- Netbeans for Java development
- soapUI plugin for Netbeans for testing of messages
- Glassfish Application Server for deployment of services

To set up a development environment with the toolchain, please follow the installation instruction provided in Chapter 4.

Figure 41 shows an overview of the MPOWER toolchain, including the tools and the most important artefacts involved in the service specification, implementation and deployment process. The Enterprise Architect tool is used to model the services in a UML class model, based on the description of the user scenarios and use-case models. From the UML model of the services, we generate a WSDL model in Enterprise Architect. The WSDL model is further transformed to a WSDL file, which is used as input for the service implementation.

From Netbeans, the WSDL file is imported and is used to generate the web service files that are needed for the implementation of the web service. When the developer has completed the implementation Netbeans is used to build the service and get the WAR file that contains all the class files. This WAR file can be deployed on an application server to make the service available for testing and for use by other services and applications.

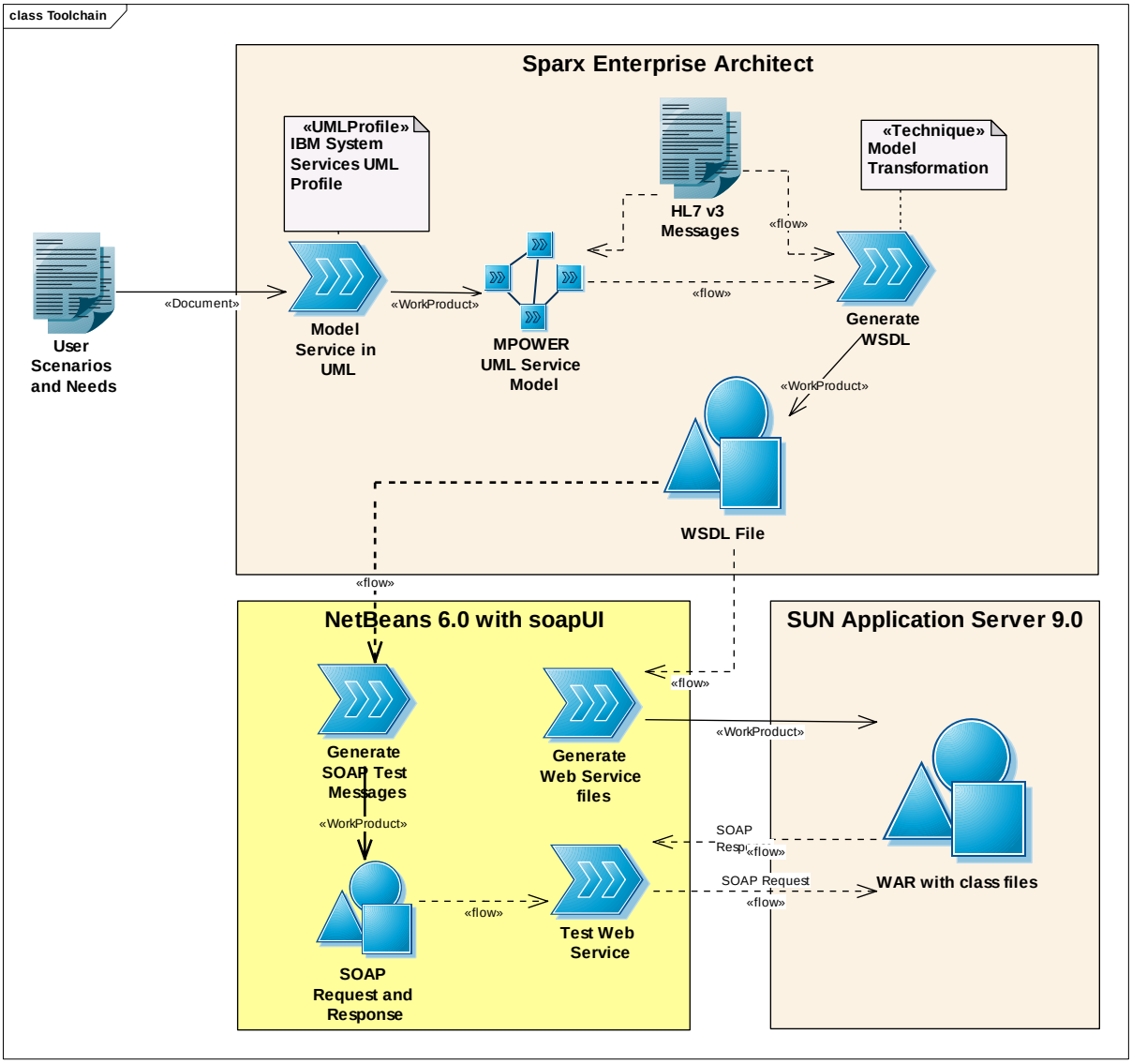


**Figure 41: Toolchain**

## *6.2.2 Information related service*

This section explains the principles of how to use the MPOWER toolchain, including which models and information that is required for the model transformations to work. Please not that section 6.3 provides further screen-shots and explanations which especially may be useful when going through the process the first time.

### 6.2.2.1 Modelling using Enterprise Architect

The recommended modelling methodology for MPOWER service development was introduced in section 6.1. This section focuses on details of how to apply the IBM SOA UML Profile using the Enterprise Architect tool, and on how to create the models required when using the MDA features of the MPOWER toolchain.

As described in section 6.1.4, the recommended process when developing MPOWER services is start with modelling of use cases and features based on user scenarios. This process uses regular UML diagrams and stereotypes, and the only tool-specific issue here is the support provided by Enterprise

Architect for specifying the traceability from use cases back to the scenario descriptions. It is illustrated on Figure 42. A use case can be derived from more than one scenario.

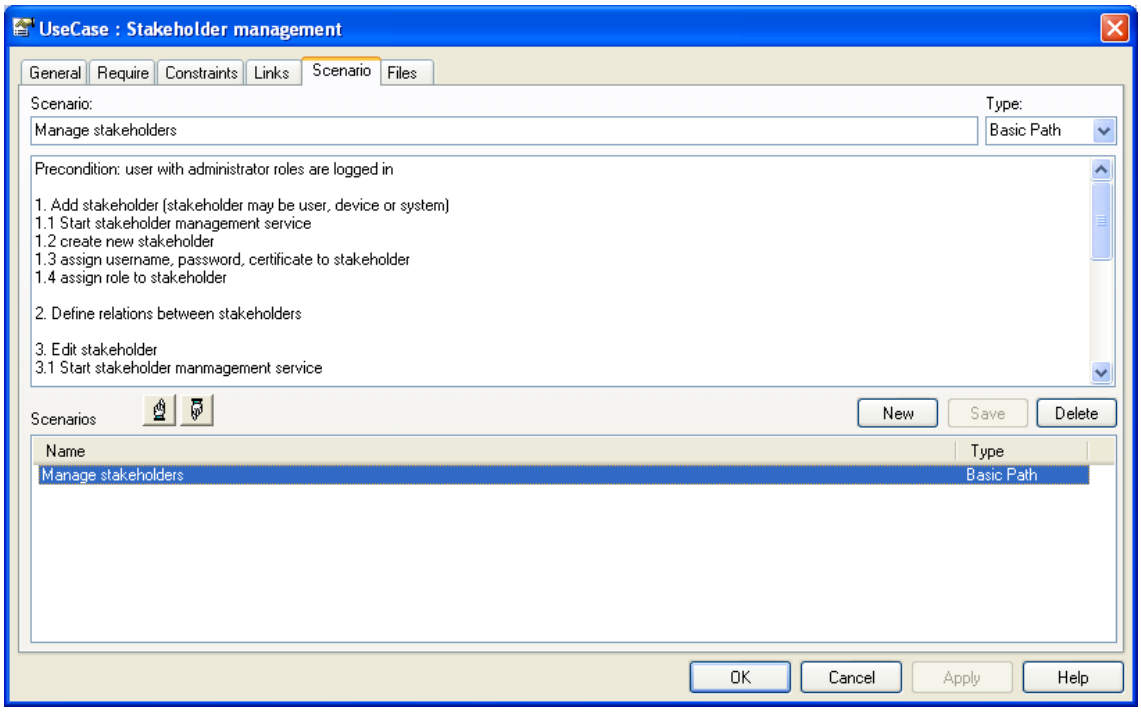


**Figure 42: Traceability from use case back to scenario**

For the service specification part of the modelling, the following set of stereotypes from the IBM SOA UML Profile should be applied:

- <<ServiceProvider>>
- <<Service>>
- <<ServiceSpecification>>

The use of these stereotypes is illustrated in Figure 43. The ActorControl applies the <<ServiceProvider>> stereotype to the UML Class. ActorControl provides the ActorManagement service, shown as a UML Port applying the <<Service>> stereotype. The ActorManagement service realizes the IActorManagement service specification, which is defined using a UML interface with the <<ServiceSpecification>> stereotype. The operations on the interface reflects the features that the service implements.

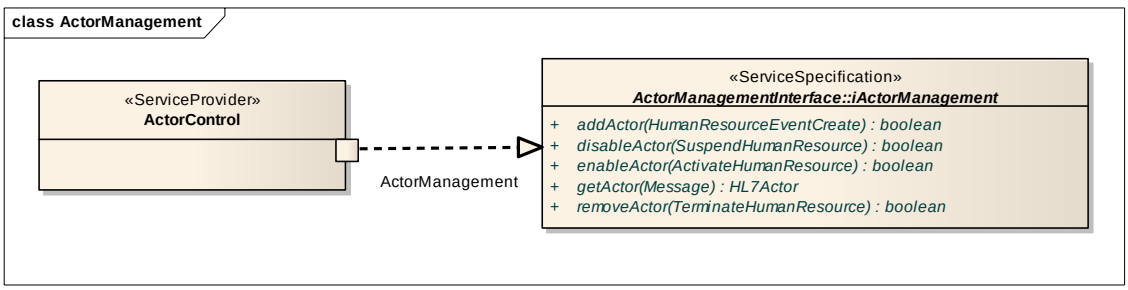


**Figure 43: The Service Model with UML Stereotypes**

### 6.2.2.2 Transforming to WSDL Model and further to WSDL

In the MPOWER MDD approach, the service model is the primary model which the toolchain will use to generate parts of the service implementation. EA supports transformation of UML class models to WSDL models. The default transformation builds elements for WSDL components, services (ports), bindings, and port-types. The MPOWER toolchain contains a customized version of this transformation which generates more information in the WSDL model than the regular WSDL transformation. See section 4.1.2.1 for install instructions.

To generate a WSDL model from the service description model in Enterprise Architect, select the package containing the service model in the project browser, and use the "Transform Current Package" action from the right-button menu. From the dialog that appears, select "include child packages" and press the "do transform" button to start. This transformation will generate a new WSDL model in the Enterprise Architect project, which is organized based on WSDL concepts and using WSDL specific stereotypes. Figure 44 shows an example of how the structure of the WSDL model will appear in the project browser. To visualize the complete service structure you may want to add the elements in Bindings, PortTypes and Services into the root-diagram of the WSDL model by dragging the elements into the diagram, producing a diagram similar to Figure 45, but this visualization is not required for tool purposes.

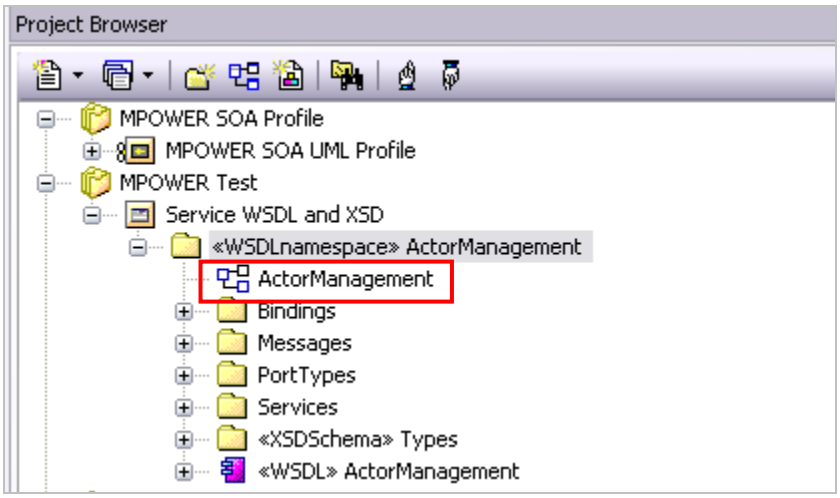


**Figure 44: WSDL model structure**

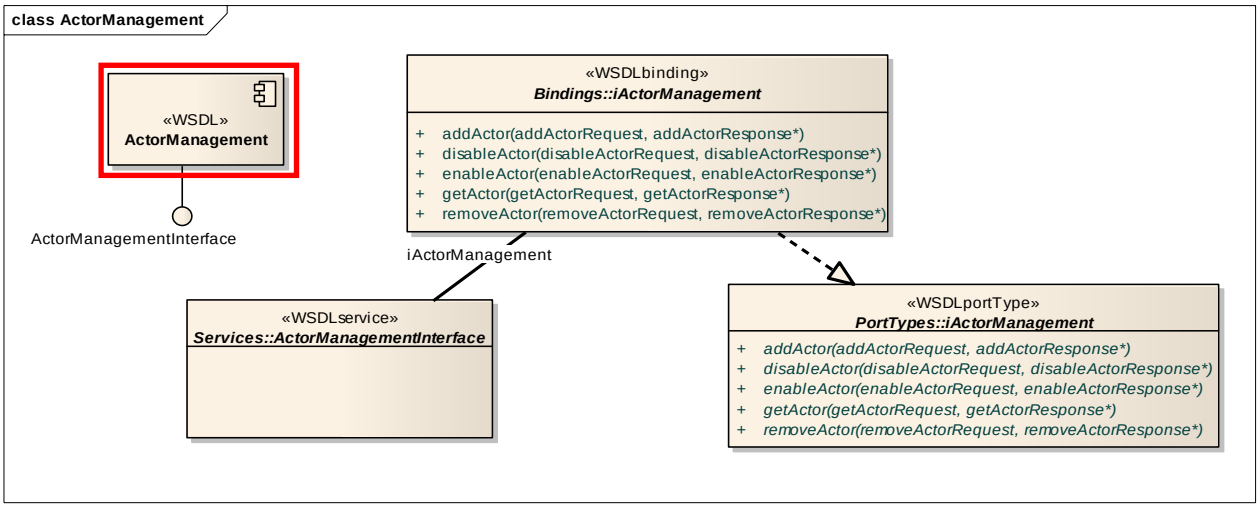


**Figure 45: From generated WSDL model with port type and binding to service**

The WSDL model resulting from the transformation is almost complete, but before WSDL code generation can be performed the target folder for the generated WSDL file must be set. This is done by setting the file name property from the properties dialog for the main component of the generated service, which is stereotyped as <<WSDL>> in the WSDL model.

To generate the WSDL file, select the main component, and choose “Generate WSDL” from the right-button menu. From the Generate WSDL dialogue which appears, select “Generate”. The result is the WSDL code generation with the WSDL file being stored to the target folder selected earlier.

Note that once a WSDL model has been generated, later transformations from service to WSDL model will update the existing model attempting to maintain any manual changes which has previously been done to the WSDL model. In general this functionality is useful, and e.g. the file name property only has to be entered once. However, if problems are experienced with the results of the transformation, a last resort can be to delete the WSDL model so that it is generated from scratch from the service model. Any previous editing such as setting the output file name or creating diagrams presentation purposes will in this case have to be repeated manually.

#### 6.2.2.3 Importing WSDL and completing the code in Netbeans

The WSDL file is a central artefact in the development of web services as it provides a definition of the interface for the service. WSDL-files can be used in most tools supporting web-service development, both when creating an implementation of the service and when using the service from an application or from another service.

In Netbeans, which is the recommended Java IDE in the MPOWER toolchain, the WSDL file is imported, and from this Netbeans will generate server stub code, the web service files that are needed for the implementation of the web service, using the JAXB and the JAX-WS specifications. When starting the service implementation from Netbeans, first create a new “Web Application” project which is found in the “Web” category from the new project dialog. To add a web service based on the generated WSDL to this project, select the web application project in the project browser, and choose “Web service from WSDL” under “New…” from the right button menu. From the dialog that appears, selected the generated WSDL file, and fill in the rest of the information requested. Netbeans will generate a skeleton implementation for the service interface in the WSDL, and will present a view of the service similar to Figure 46.

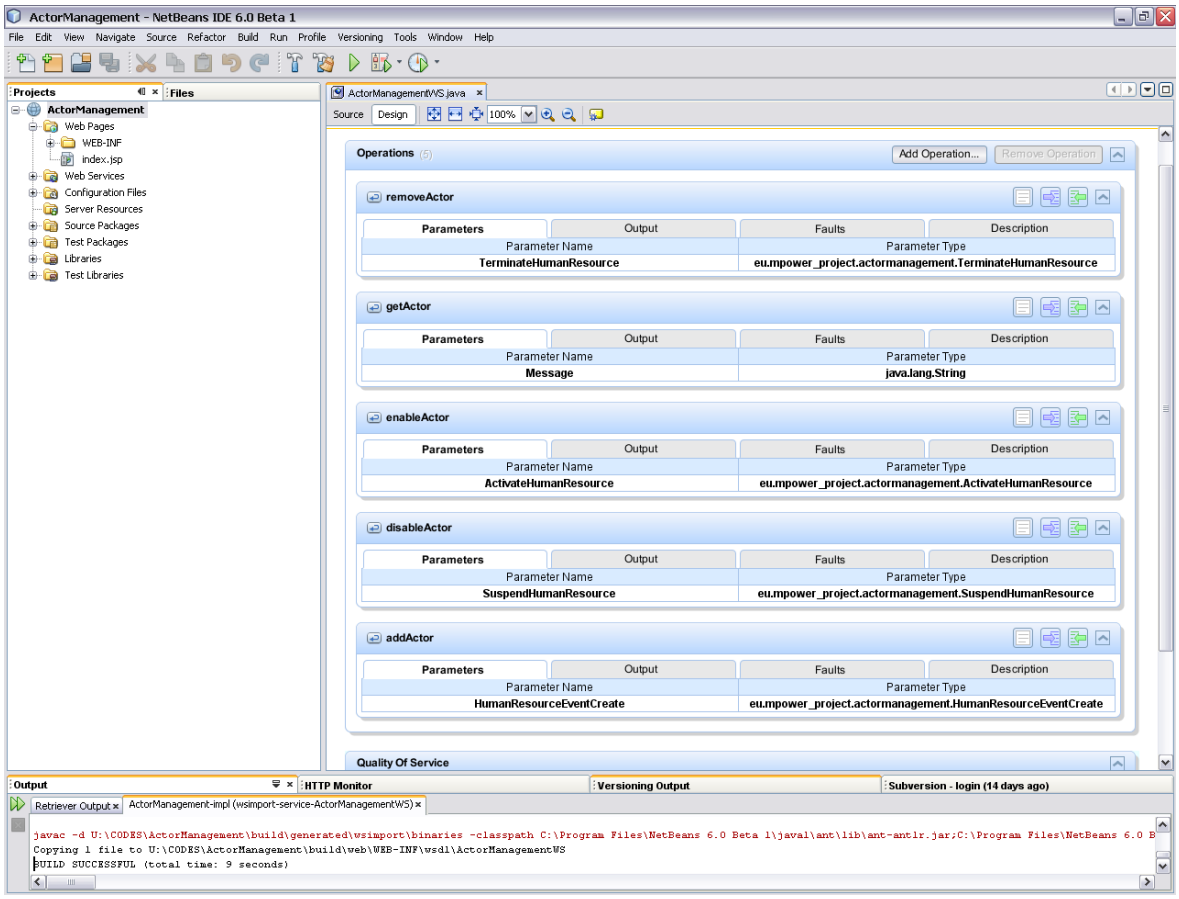


**Figure 46: Generating web service**

The skeletal implementation of the service is a Java class with web service annotations. To view the Java source code, select “Source” instead of “Design” in the top left corner of the editor. The generated Java class will contain a method skeleton for each of the operations in the web service. To complete the implementation of the web service, these methods must be implemented. The methods will typically access or modify content of a database based on the input parameters to the method, and will often return a result. Most of the current MPOWER services use Hibernate to implement the database operations, but it is up to the developer of new services to select the approach to use for this.

### 6.2.2.4 Deploying from Netbeans

When you finish with the implementation you can right click on the web application project to build the project and get the WAR file that contains all the class files (Figure 65). Now you only need to deploy the service to an application server and this can be done by right clicking the project and choosing “Deploy” option (Figure 66). If your web service successfully deploys you will get an address on which web service end point is listening (Figure 67).

The WAR file can be deployed on any server to make the service available to those who have the permissions to access it.

### 6.2.2.5 Testing with soapUI

The soapUI plugin to Netbeans is a tool for testing your deployed web services. The tool allows the developer to test a deployed service by creating SOAP request and receiving SOAP responses.

To perform testing with soapUI from Netbeans, you first have to create a Netbeans project using the “Web Service Testing Project” type which is available under the SOA project category from the “New Project” wizard when the soapUI plugin have been installed. The wizard of the testing project will

allow you to select the WSDL of the service to test. Check the “Generate TestSuite” check box in the wizard. Further description of how to use soapUI can be found in the soapUI user guide:

- http://www.soapui.org/userguide/index.html

Further description the integration of soapUI with Netbeans can be found at:

- http://www.soapui.org/Netbeans/index.html

### *6.2.3 Physical level related services and components*

MPOWER platform is an open platform, ready to be improved by adding new services. Different kind of services can be added. E.g. installation of a new type of sensor/device in the environment can result in adding services for accessing that sensor/device.

When a new sensor is to be integrated in the platform it can be done in five different manners. Each one of them requires a specific work to be done when setting up the new service. Aforementioned FSA is in charge of offering a homogeneous way of accessing the real heterogeneous network of sensors within the environment of the MPOWER platform has. The architecture of FSA is divided into three layers: Device Service Layer, Virtual Sensor/Actuator Layer and Adapter Layer. It is important to take that into consideration and follow the steps presented on the figure below to process the task of creating a new service.

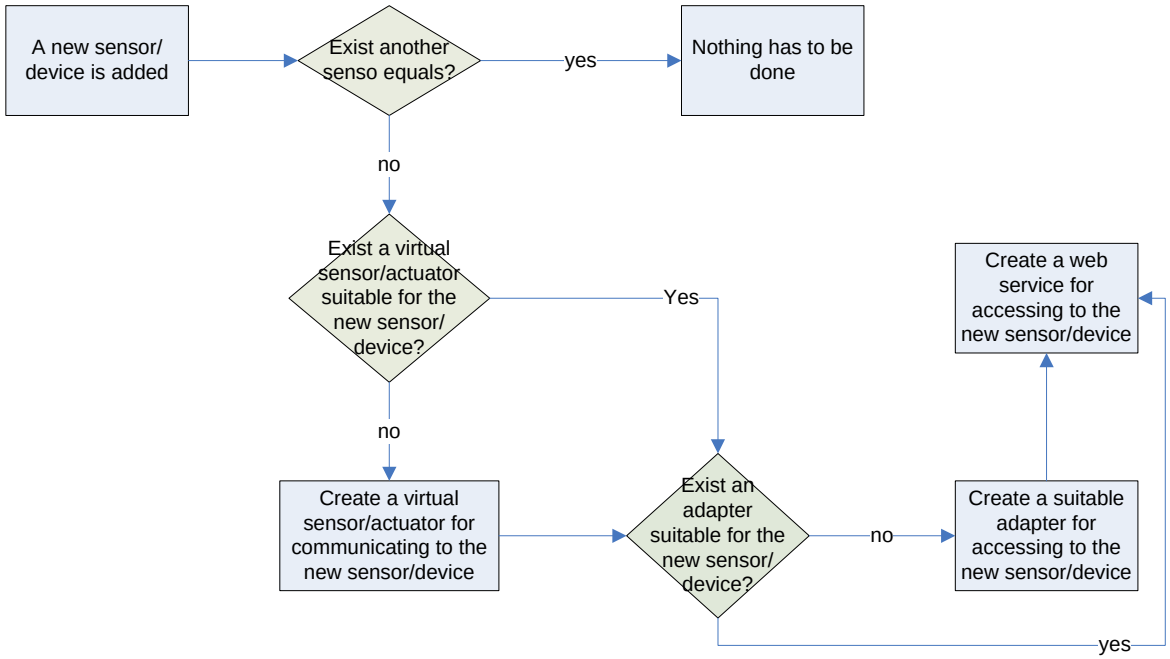


**Figure 47: Steps to be followed when creating a new service for accessing a new device type**

Depending on the actual components and services deployed to the platform different steps should be executed for creation of a new service for device accessing.

First of all checking on already existing sensor/device of the same type should be executed. In case it exists new service does not have to be created.

If this is not the case and no existing service can be used then first question to be answered is “Does there exist a virtual sensor/actuator suitable for the new sensor/device?” That means, does there exist

some component in the virtual sensor/actuator layer that knows to interpret the information that new sensor is going to send? If the response is positive execute the next step. In case of negative response, a new virtual sensor/actuator component able to interpret the information coming from the new sensor should be created. For creating a new sensor/actuator it is necessary to know the specification of sensor/device provided by manufacturer to code a component enable to handle the communication with sensor/device according to the given specification.

In the next step the question to answer is “Does there exist an adapter suitable for the new sensor/device?” In other words, does there exist an adapter able to deal with communication channel for the new sensor/device and the information that is going to be sent? In case of positive response the process can be continued in the next step. In case of negative response, new adapter component able to deal with communication channel for the new sensor should be created. The created adapter component must follow requirements specified (protocol) by communication channel that it is going to deal with.

When the process finishes an interface via which applications MPOWER-based could access the new sensor has to be created. That interface is a web service which is in charge of providing a single access that is offered by FSA for communicating with sensors/devices.

### *6.2.4 HL7 particular case*

As part of the medical and social services of the MPOWER platform the Medication management and Calendar management services have been developed. We choose to describe these services as an example of the overall service process definition in MPOWER platform. Additionally, we choose these service as example since they involve HL7 health standard, thus we can demonstrate the inclusion of health-standards into service design process.

Health Level Seven is an American National Standards Institute accredited Standards Developing Organizations operating in health-care area. Health Level Seven’s domain is clinical and administrative data, which by definition maps well to the MPOWER domain scope. HL7 Version 3 standard [8] uses well-defined methodology based on a Reference information model. The reference information model provides an explicit representation of the semantic and lexical connections that exist between the information contained in HL7 messages that are used to exchange medical data. The HL7 methodology is fully technology independent, thus the process of going from business analyses until completely defined message contents uses UML modelling paradigm and related diagrams, and is completely independent of any technology that will be used in implementation stage. However, most of the systems, including MPOWER, use XML enabled technology as the final message container. The requirement of respecting the methodology implies that messages need to be specified using the reference information model as the starting point. Further, messages are refined through domain information model that select and defines messages that are relevant to some medical domain to a refined message information model. The refined message information model contains messages descriptions for specific operations and services. From this point HL7 tooling is used to derive serialized messages and XML schemas that define the form and the content of the message in a computer processable format.

#### 6.2.4.1 Service definition process

The overall process of defining information models in the MPOWER platform is presented in Figure 28. The process uses “top-down” approach, thus it starts from detailed business requirements and refines them in a stepwise fashion down to a software implementation. In the process, we identify the required services, define interfaces that form the services, define the operations that make interfaces, and finally define the messages that are used in operations. In the figure we can see parallel modelling

sequences of HL7 and MPOWER methodologies. The parallel modelling sequences are required because each step of MPOWER modelling needs to be synchronized and correlated to HL7 specification in order to finally arrive to a complete service definition that is conformant to HL7 standards.

## 6.3 Step by Step guide for creating a service

***Note to the reader: the content of this sub-chapter do to some degree overlap with the content of the previous sub-chapters. However, the content of this chapter can still be of use to you, but some issues are covered in less detail than one might expect, also the example provided is somewhat larger than what one expected as a tutorial.***

Following the discussion in Chapter 6.1, to run the service modelling process must have a working installation of Enterprise Architect (Corporate edition 6.5.x) from Sparx Systems. In addition, for generating server stub code based on web service description language document and deploying the newly generated service should download Java EE 5 SDK Update 3 (this bundle includes Netbeans IDE 6.0 Beta 1 and Sun Java System Application Server 9.1)

[http://java.sun.com/javaee/downloads/ea/]. For testing the service operation with basic input and output parameters, Netbeans can be used.

### *6.3.1 Service modelling*

Here we will start from the point where the user scenarios are modelled in UML use cases. In Figure 48, is shown a part of the management UML use case diagram that shows the stakeholder management use case which maps to specific scenarios of use (Figure 49). Then from the user scenarios and the use cases we can extract the features (non functional and functional). The functional features are shown in Figure 50 and indicate some “services” that need to be implemented. The service ActorControl realizes those features as shown in Figure 51. So after we decided the service that will be implemented we need to specify the service interface as shown in Figure 52. The services should be modelled according to SOA Principles described by Erl, and guidelines on using IBM System Service UML Profile.

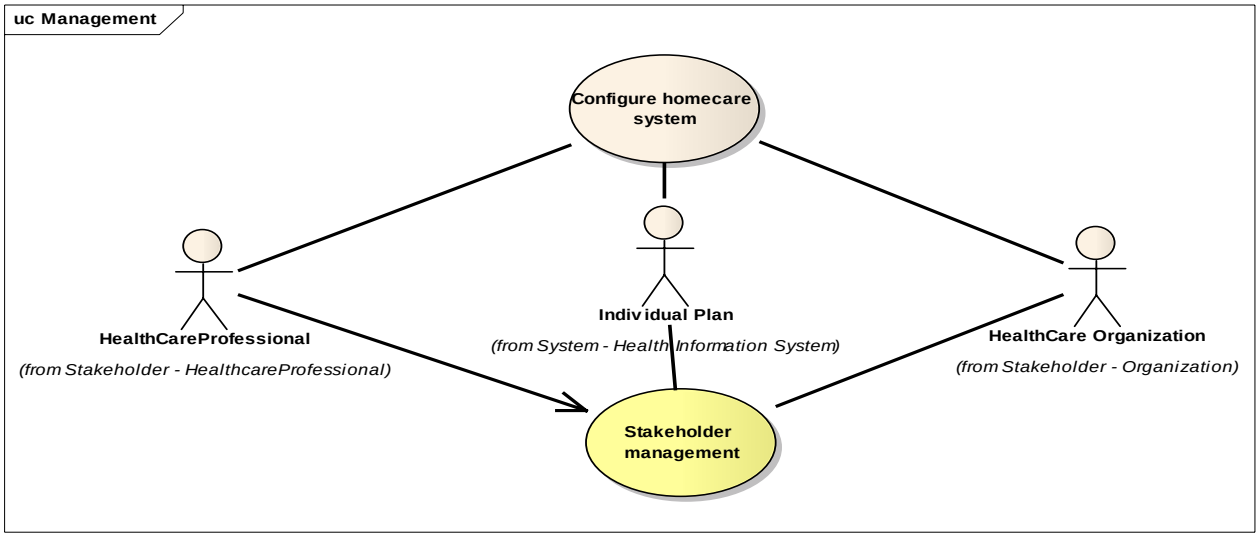


**Figure 48: Management use case diagram**

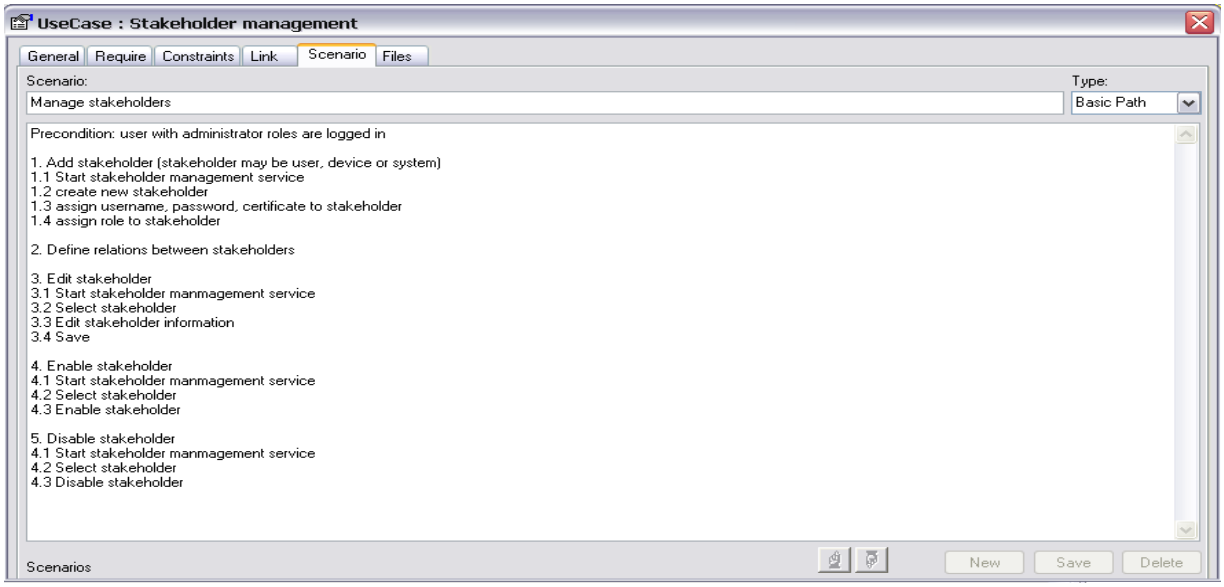


**Figure 49: Use case - Scenario mapping**

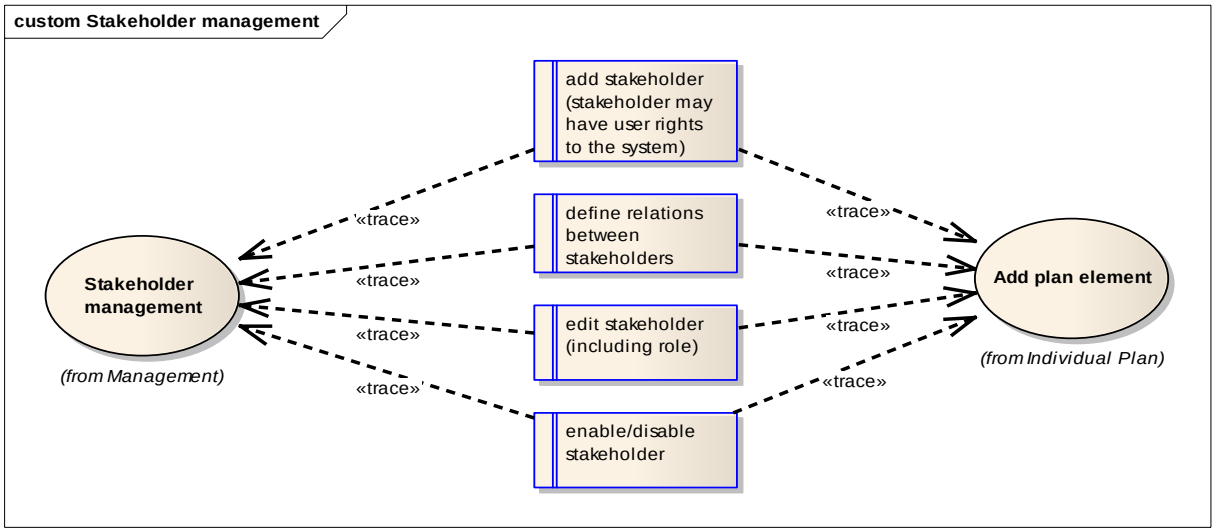


**Figure 50: Stakeholder management features**

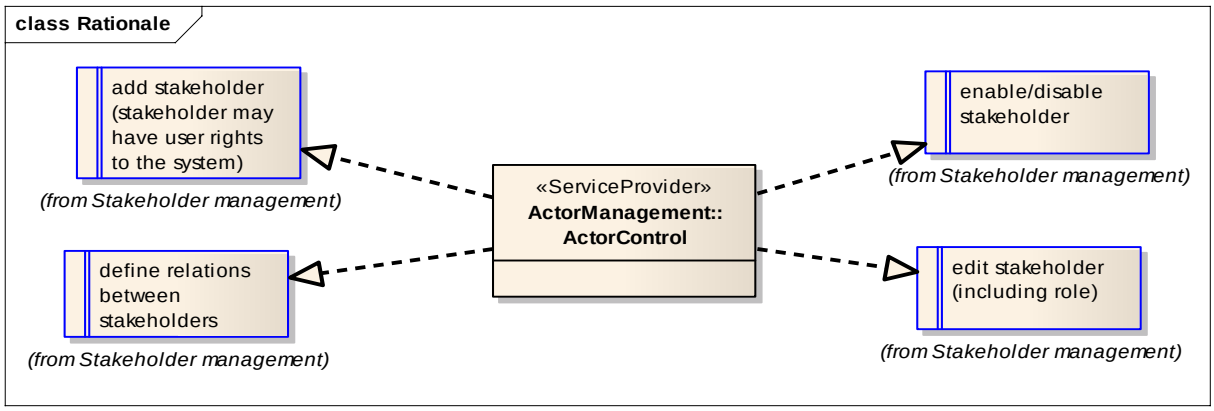


**Figure 51: Actor Management rational**

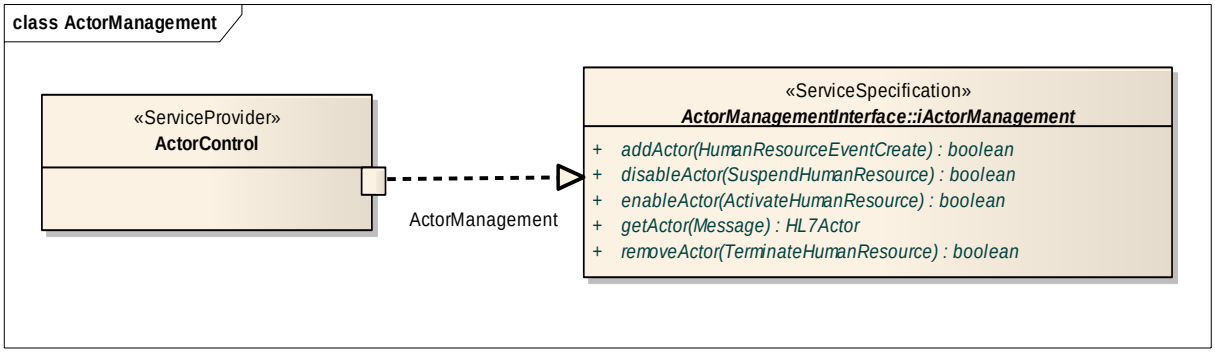


**Figure 52: Actor Management service interface**

### *6.3.2 WSDL model transformation*

Following the MDD approach we use the models as primary artifacts throughout the engineering lifecycle. EA supports transformation of UML class models to WSDL models. The default transformation builds elements for WSDL components, services (ports), bindings, and port-types. The MPOWER WSDL transformation template (called MPowerWSDLTemplate.xml which you will find in: https://project.sintef.no/eRoom/informatics/MPOWER/0_5037b) will generate more information of the WSDL than the regular WSDL transformation. To use the template, you first have to import it into the Enterprise Architect project. This is done by choosing the "Import reference data" from the "Tools" tab. This import will modify some of the built-in transformations for WSDL. So the next step is to use the MPOWER WSDL transformation template to generate a WSDL model from the service description model in Enterprise Architect (as described below).

The first step is to right click on the <*servicename*> package and choose Transform Current Package (Figure 53). Then, in the dialogue that appears, check the WSDL box and choose an appropriate target location. Make sure to include child packages and press the “Do Transform” button (Figure 54). When the process is done, a new WSDL package has been created in the <*target package*> (Figure 55). To visualize the complete service structure, double-click the root-diagram to open (here: ActorManagement class diagram, red rectangle in Figure 55) and add (drag) the elements in Bindings, PortTypes and Services onto the root-diagram (Figure 56).

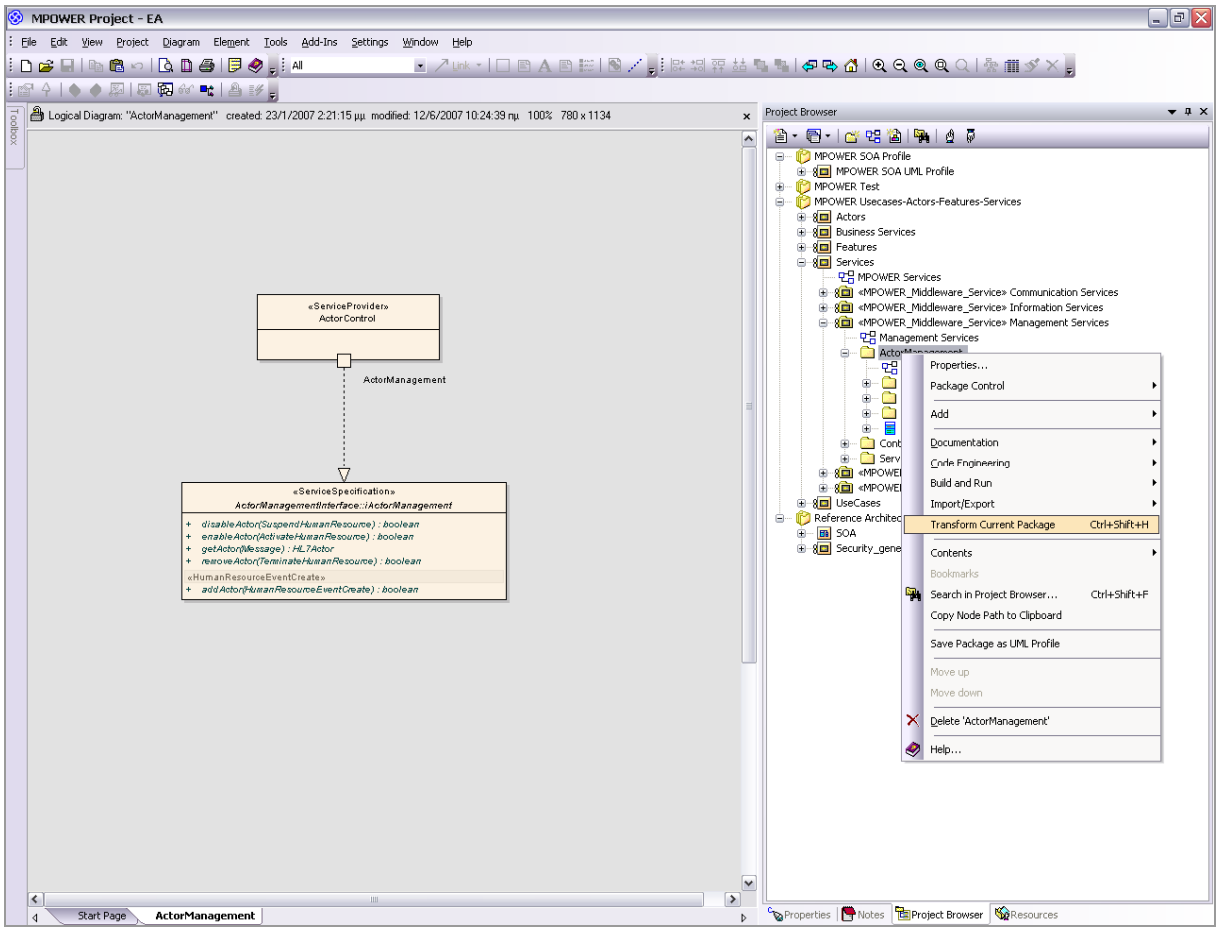


**Figure 53: Transformation a Service Package to WSDL in EA**

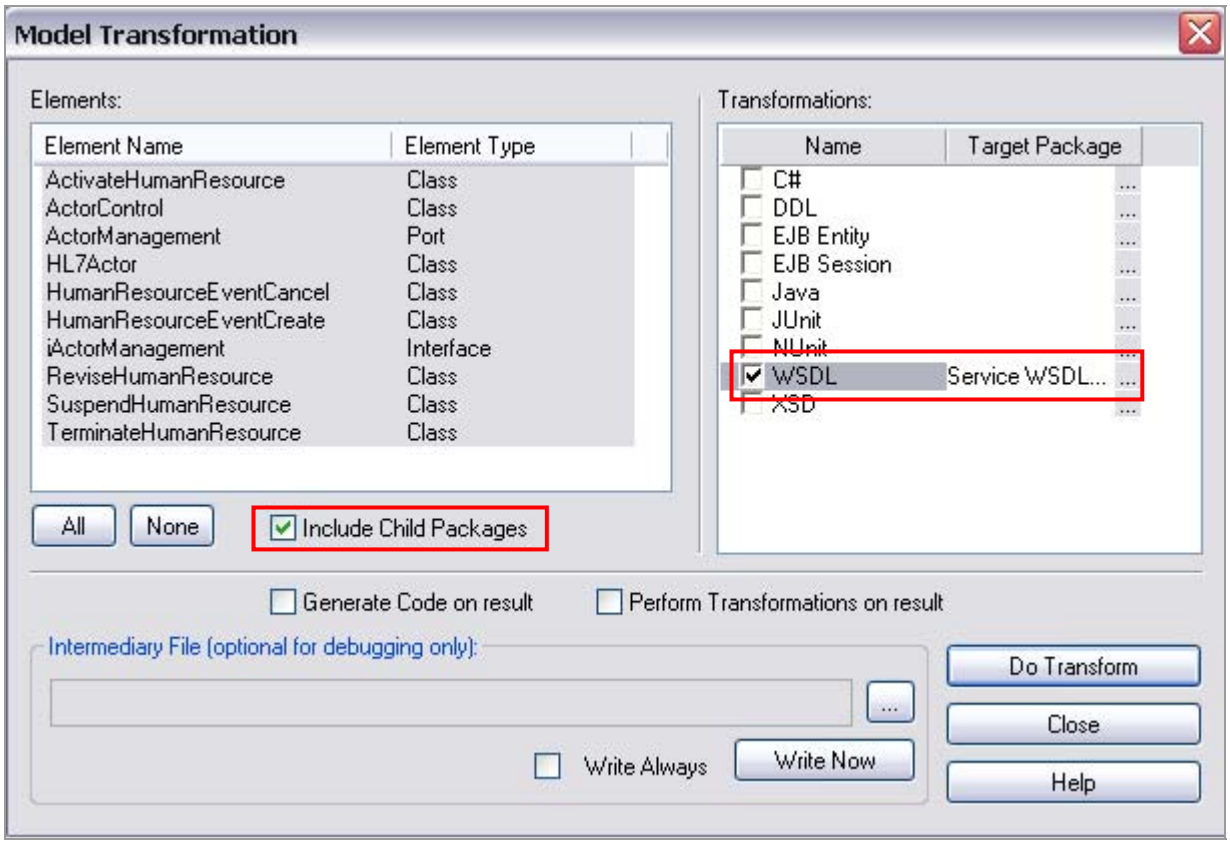


**Figure 54: The Model Transformation Dialogue**

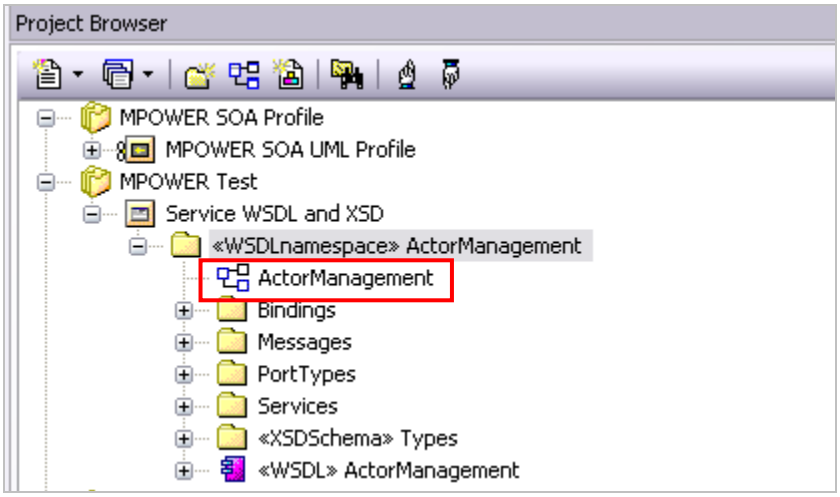


**Figure 55: The transformation WSDL model hierarchy**

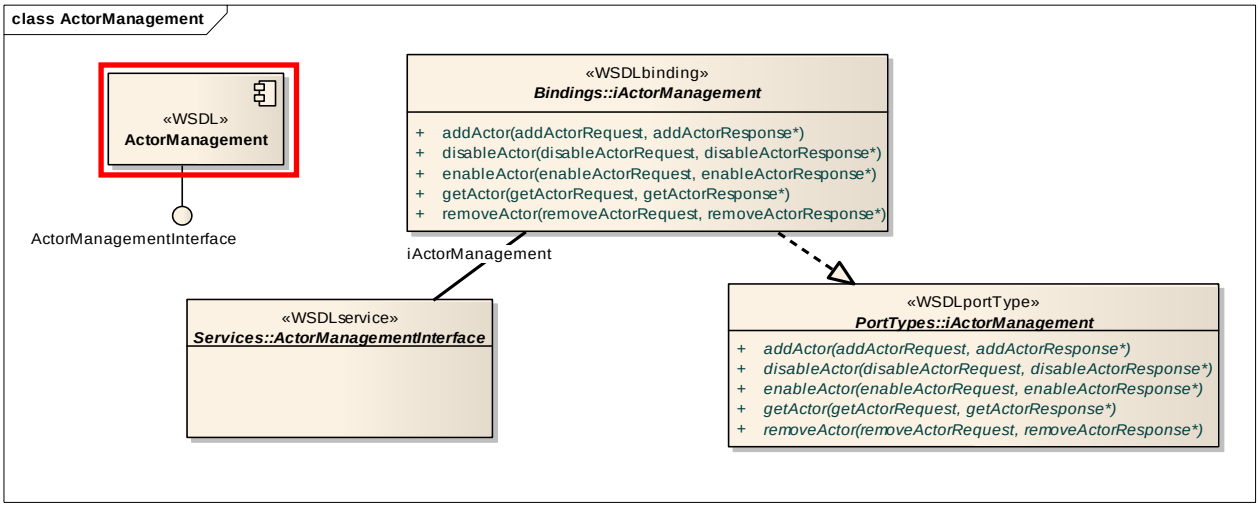


**Figure 56: Complete service structure, with port type definition and binding to service**

As a preparation for the next step that is the WSDL code generation, to set the target folder for the WSDL file, double-click the WSDL component (red rectangle in Figure 55) in the root diagram (here: ActorManagement class diagram, red rectangle in Figure 56) and specify the WSDL filename (the service name can be used) in the second text-field in the dialogue.

To generate the WSDL file right-click the WSDL component and choose Generate WSDL (Figure 57). The Generate WSDL dialogue pops up (Figure 58). EA uses UTF-16 encoding whereas Netbeans uses UTF-8 encoding as default. UTF-16 may work, but we choose to change into UTF-8. Do this change and press “Generate”. The result is the WSDL code generation with the WSDL file being stored to the target folder we selected earlier.

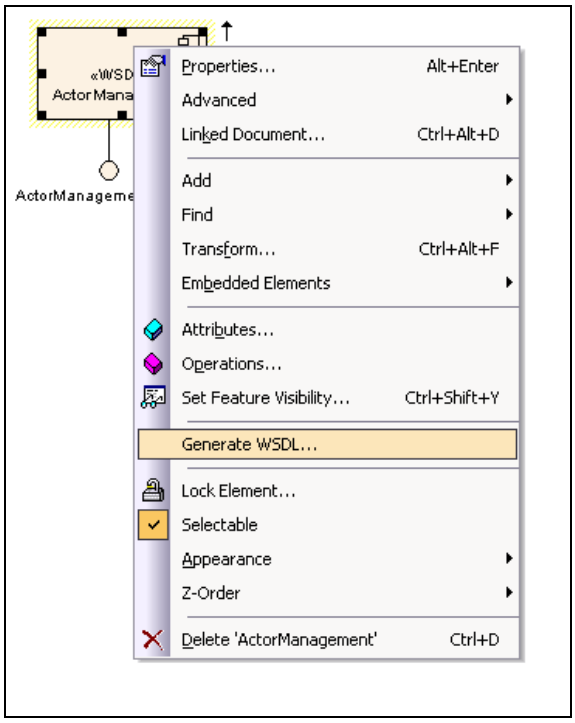


**Figure 57: WSDL file generation**

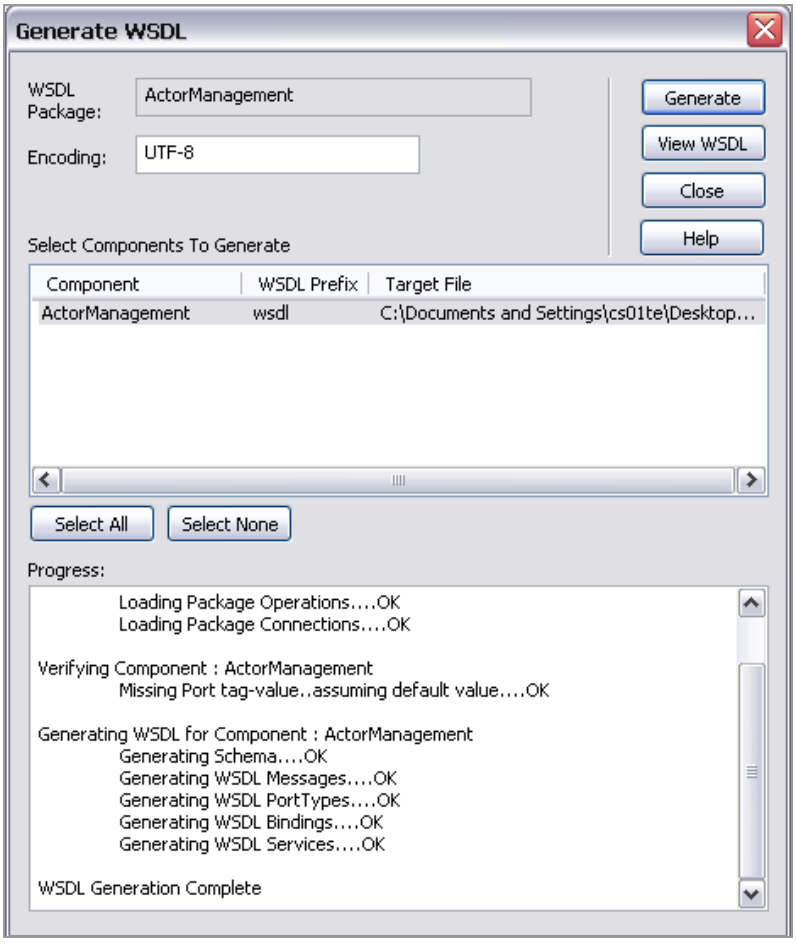


**Figure 58: Generate WSDL dialogue**

WSDL manual editing

### *6.3.3 Service Implementation*

The implementation phase starts with the wsdl generation from Sparx Enterprise Architect. This wsdl file is used as an input to Netbeans IDE. Netbeans generates server stub code, the web service files that are needed for the implementation of the web service, using JAXB and JAX-WS specifications.

The first thing to do is create a new Netbeans project by choosing “New Project” from the “File” tab. This command opens a dialogue on which you can choose the category and type of project you want to create. For this purpose, should choose “Web” category folder and “Web Application” project as the type (Figure 59). Then you press the “Next” button and should see a new dialogue on which you should enter basic information about your project including name and location path (Figure 60). Next you press “Finish” and the new Netbeans project named “ActorManagement” is created.

Now, to generate a new web service from the wsdl file EA produced, should right click on the created web application project (under “projects” tab) and choose New → Web Service from WSDL (Figure 61). A dialogue opens and you should write name and package for your web service as well as determine the path of wsdl file from which you want to generate the service (Figure 62). When you press the “Finish” button, code generation starts and if everything works you should see service interface implementation like the one on Figure 63 You can change to the Source view and complete the code of the service by implementing the web service interface (Figure 64).

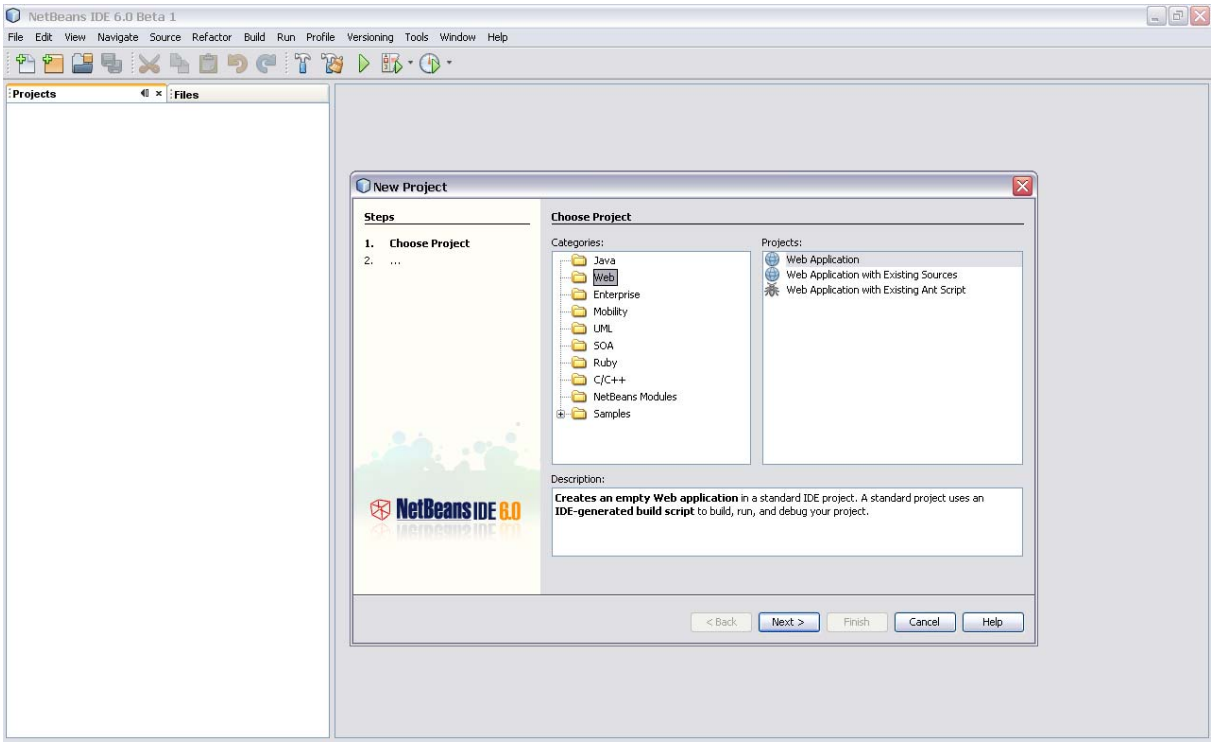


**Figure 59: Create new Netbeans project-type**

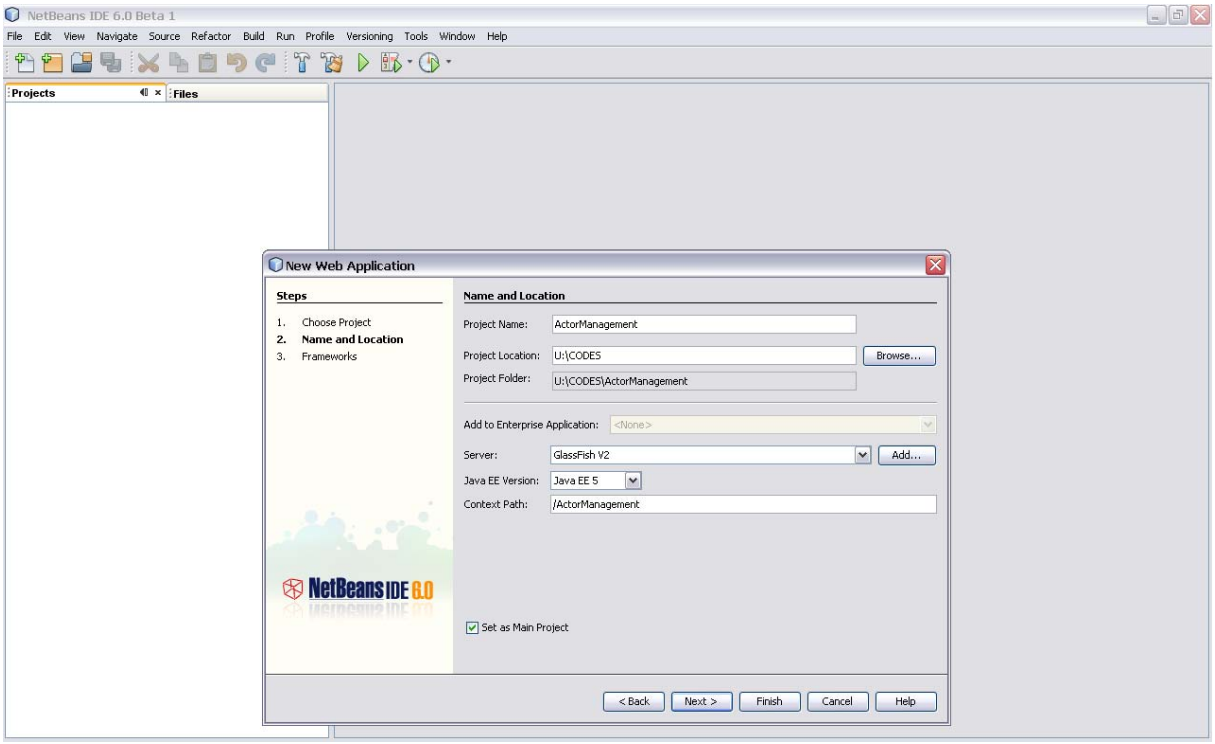


**Figure 60: Create new Netbeans project-information**

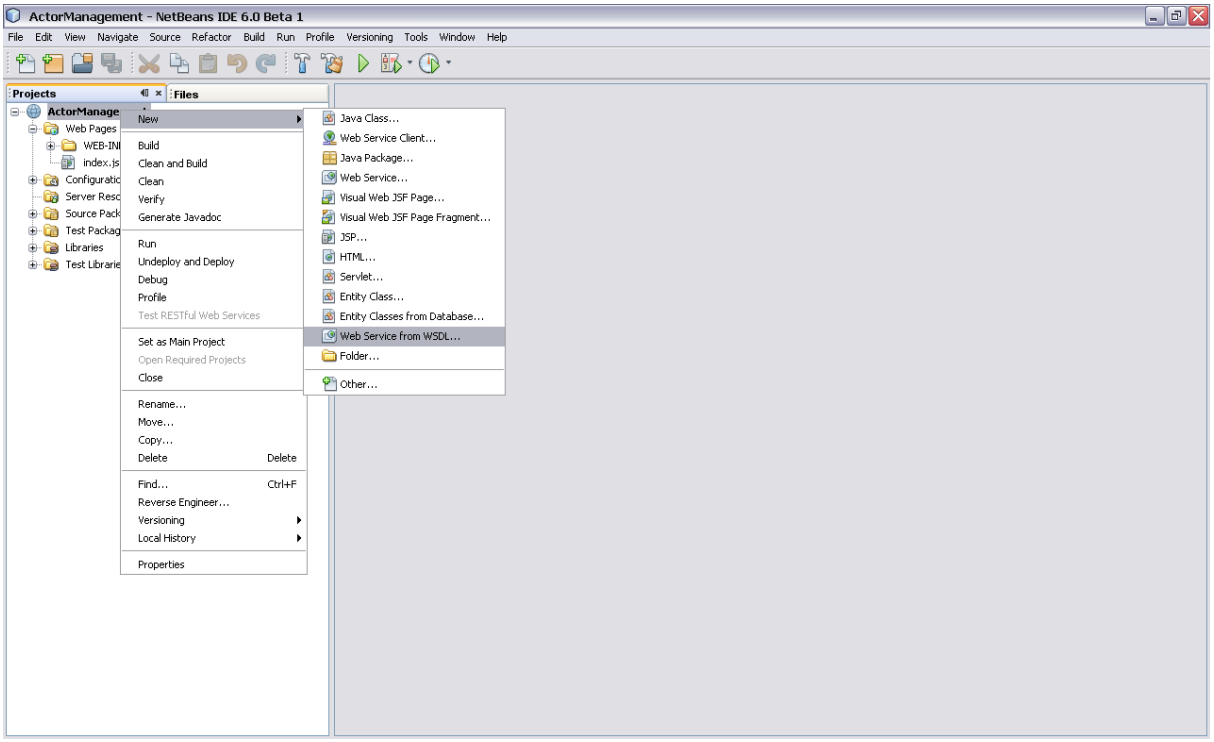


**Figure 61: Generating new service from wsdl file**

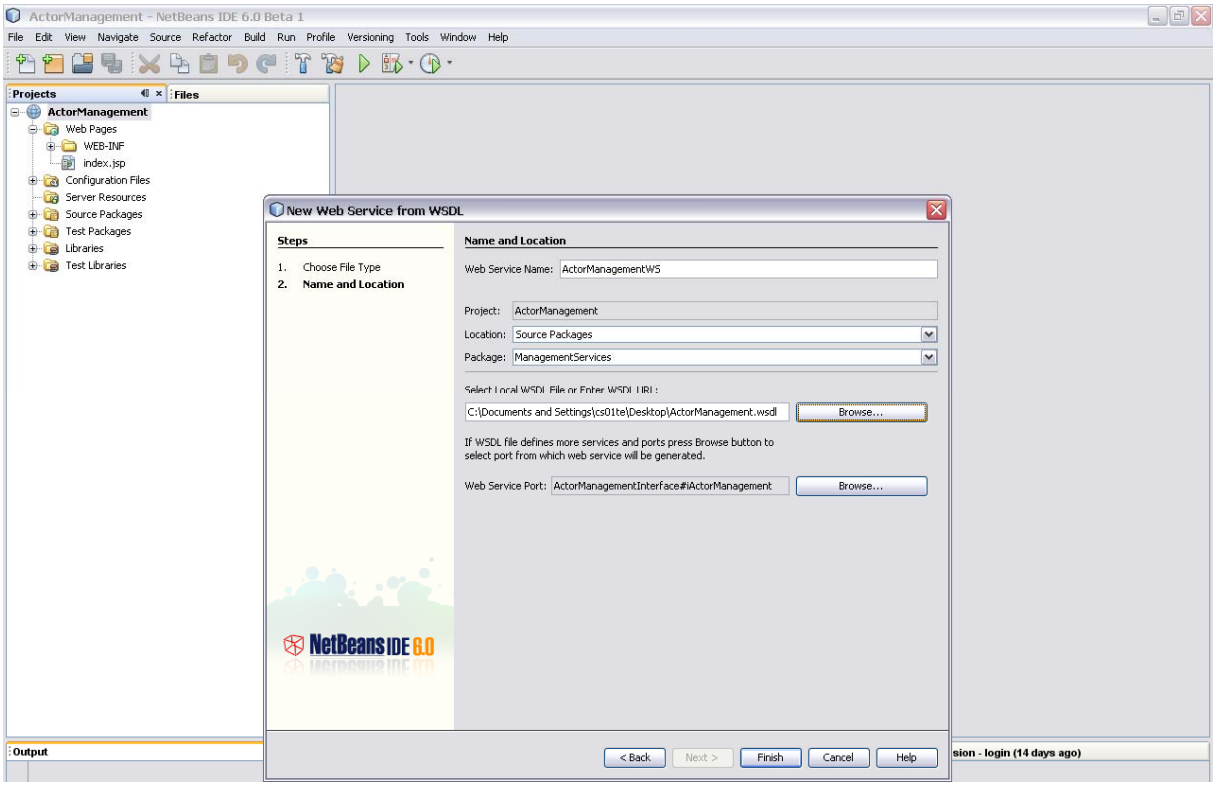


**Figure 62: Generating new service from the wsdl file-information**

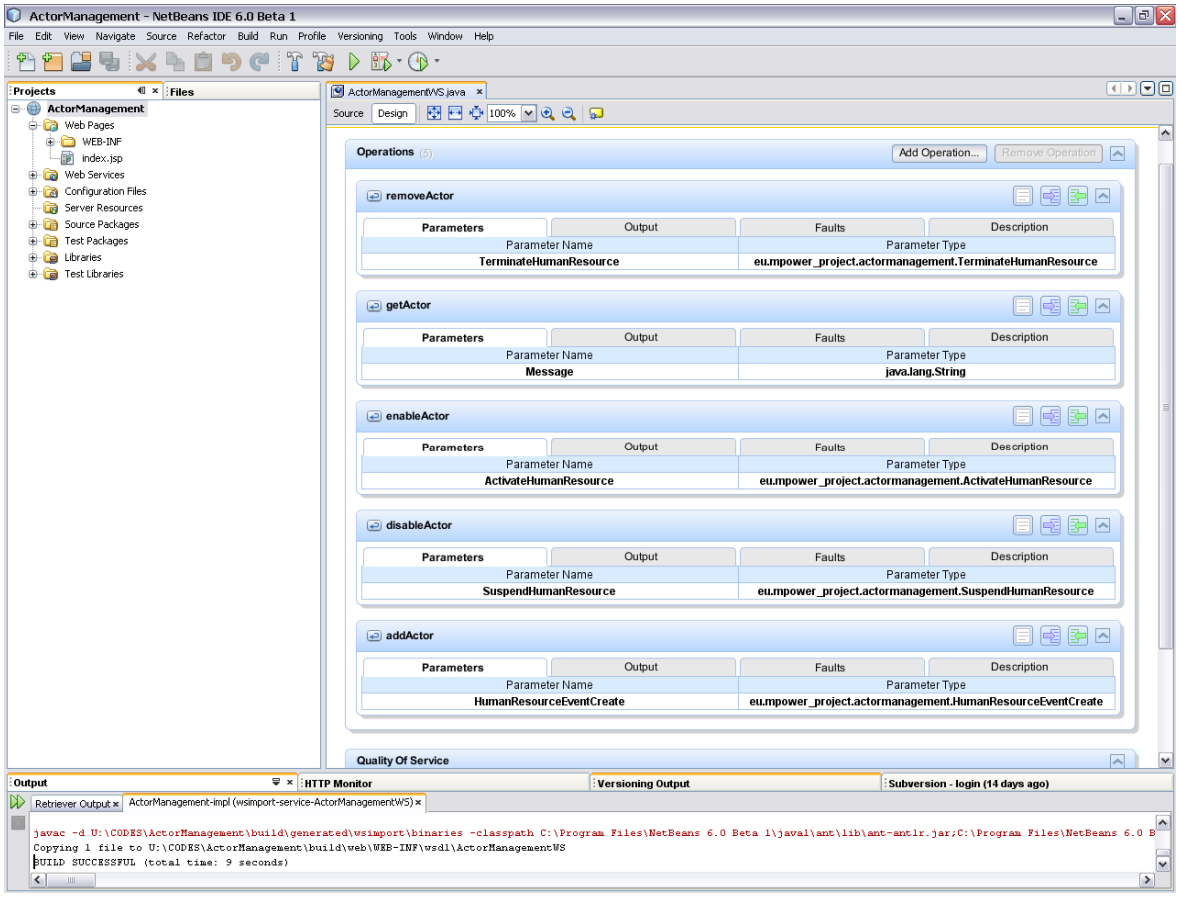


**Figure 63: Generating web service**

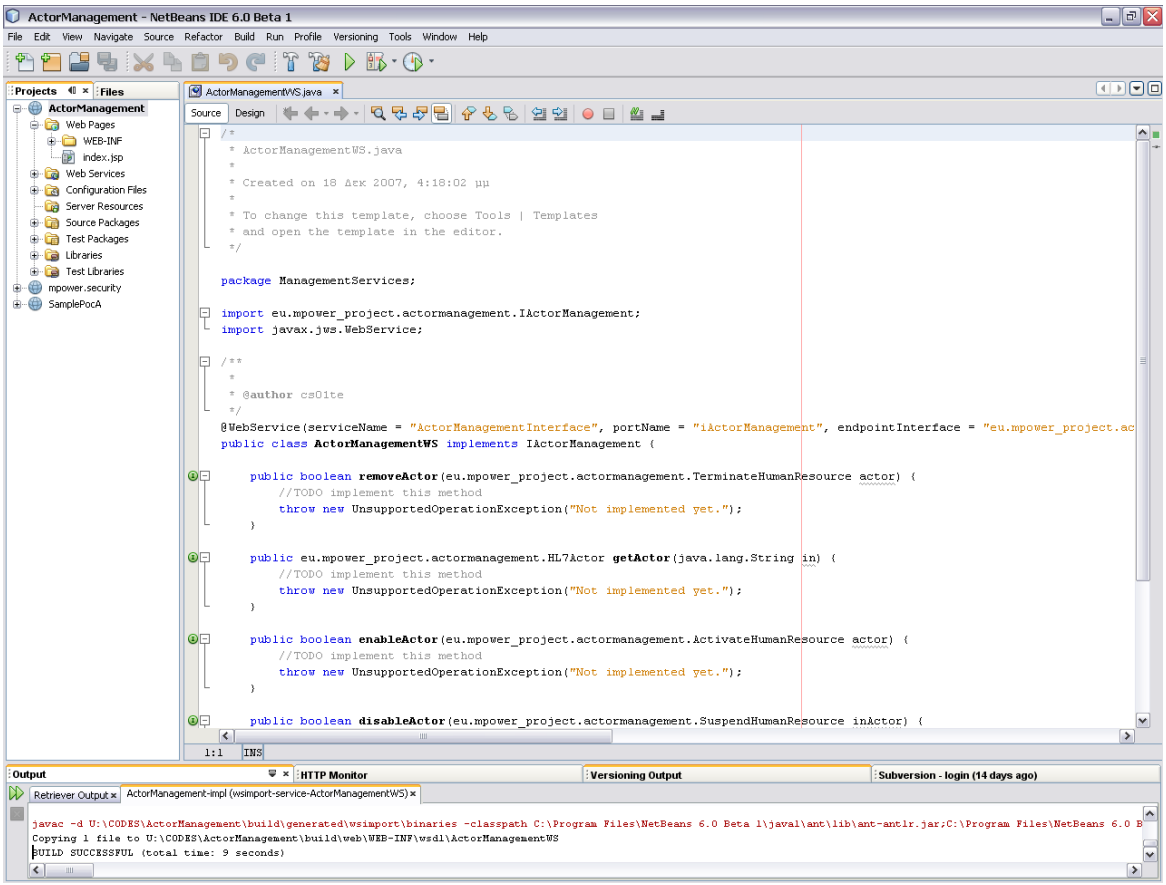


**Figure 64: Generating web service - source view**

### *6.3.4 Service Deployment*

When you finish with the implementation you can right click on the web application project to build the project and get the WAR file that contains all the class files (Figure 65). Now you only need to deploy the service to an application server and this can be done by right clicking the project and choosing “Deploy” option (Figure 66). If your web service successfully deploys you will get an address on which web service end point is listening (Figure 67). To test if your deployed web service is working well you should send soap request to this server and get a meaningful soap response. This can be done by creating a web service client that generates and sends a soap request to the server (deployed service) and retrieves a soap response.

The WAR file can be deployed on any server to make the service available to those who have the permissions to access it.

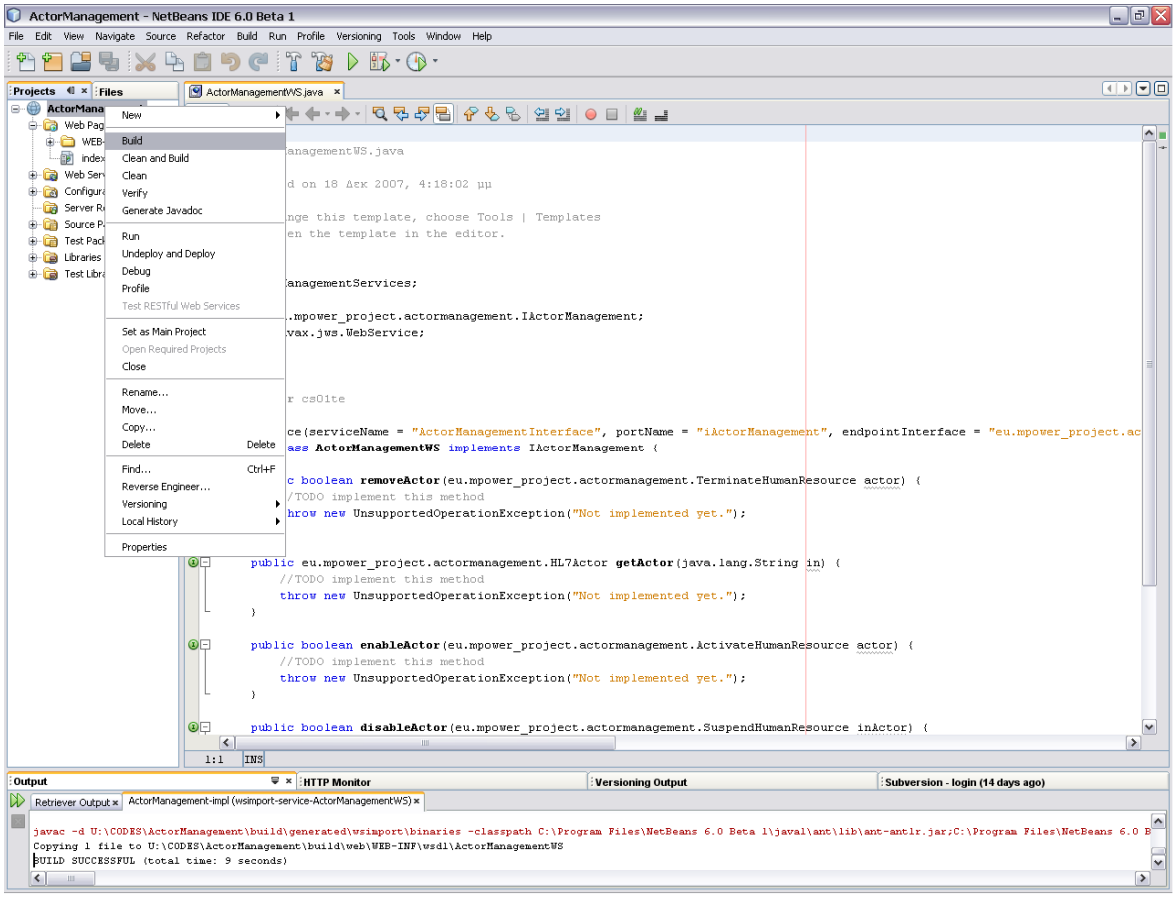


**Figure 65: Build the project**

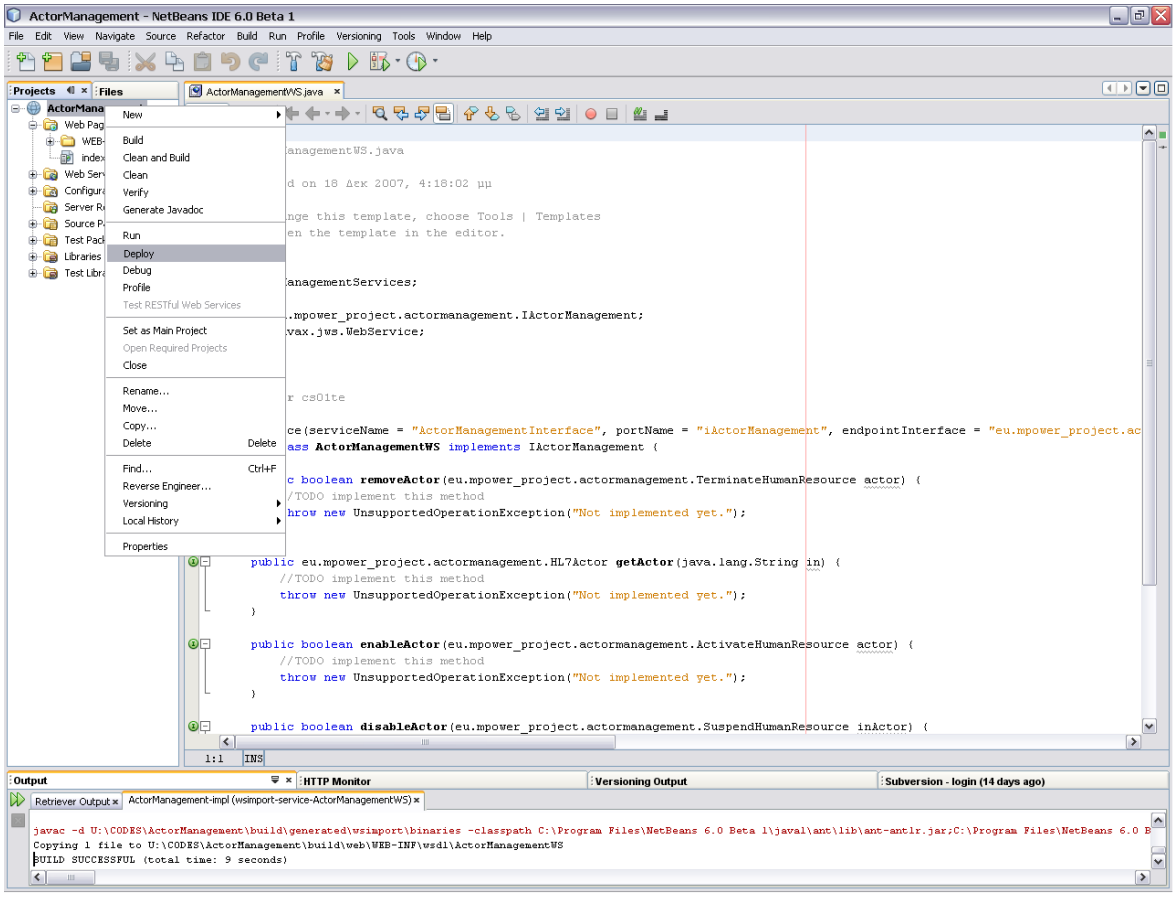


**Figure 66: Deploy the web application**

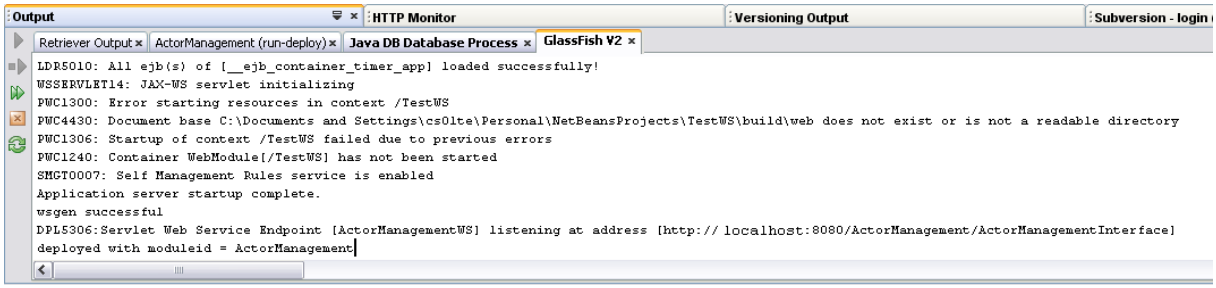


**Figure 67: Address of web service endpoint**

# Definitions, abbreviations and acronyms

**Middleware:** is computer software that connects software components or applications.

Toolchain: is the set of computer programs that are used to create a product. The tools may be used in a chain, so that the output of each tool becomes the input for the next.

**Service Lifecycle:** The SOA Lifecycle is a model that is intended to illustrate relationships and dependencies between various independent lifecycles that comprise a mature, enterprise SOA program.

**SMART HOUSE:** Smart house is a field within building automation, specializing in the specific automation requirements of private homes and in the application of automation techniques for the comfort and security of its residents.

**UDDI:** is an XML-based registry for businesses worldwide to list themselves on the Internet. UDDI is often compared to a telephone book's white, yellow, and green pages. The project allows businesses to list themselves by name, product, location, or the Web services they offer.

**ANSI:** American National Standards Institute.
**API:** Application Programming Interface.
**CIM:** Computation Independent Model.
**EA:** Enterprise Architect
**ECG:** Electrocardiogram.
**EIB:** European Installation Bus.
**ESB:** Enterprise Service Bus
**FSA:** Frame Sensor Adapter.
**HL7:** Health Level 7.
**HVAC:** Heating, Ventilation and Air Conditioning.
**IDE:** Integrated Development Environment
**JDK:** Java Development Kit
**KNX:** Konnex.
**MDA:** Model Driven Architecture.
**MDSD:** Model Driven Software Development.
**OMG:** Object Management Group.
**PIM:** Platform Independent Model.
**POCA:** Proof Of Concept Application
**PSM:** Platform Specific Model.
**QoS:** Quality of Services.
**RIM:** Reference Information Model.
**SDO:** Standards Developing Organizations.
**SOA:** Service Oriented Architecture.
**SW:** Software.
**UML:** Unified Modelling Language.
**UDDI:** Universal Description, Discovery, and Integration

# References

[1] Java web service resource
http://java.sun.com/webservices/

[2] Web service resource
http://www.webserviceresource.com/

[3] D1.1 Overall Architecture
https://project.sintef.no/eRoomReq/Files/informatics/MPOWER/0_4e96d/D1.1%20Overall%20%20Architecture.doc

[4] Judith Hurwitz, Service Oriented Architecture for Dummies , Wiley Publishing, Inc., 2007

[5] Enterprise Service Bus – OpenESB
https://open-esb.dev.java.net/AboutOpenEsb.html

[6] Rules Engine.
http://www.jboss.com/products/rules

[7] D2.2 Smart house and sensor integration design
https://project.sintef.no/eRoomReq/Files/informatics/MPOWER/0_60bbd/D2%202Smart_House_and_sensor_integration_design%20%20with%20device%20manager(2)-4.doc

[8] HL7 v3 Normative Edition 2006
http://www.hl7.org

[9] George T., Westermann G.: Model Driven Architecture in einer service-orientierten Anwendungslandschaft; JavaSpektrum 1/2006

[10] MDA – OMG
http://www.omg.org/mda/

[11] Using a Manually Created WSDL as a Web Service Client – last updated February 2008-10-24;
https://open-esb.dev.java.net/kb/v2/javaeesetut.html

[12] HL7 Reference Information Model;
http://www.hl7.org/Library/data-model/RIM/modelpage_mem.htm