

A UML Based Methodology for the Development of Web Services

An Approach to Model Transformation and Code Generation

Dissertation
zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Goerg-August-Universität zu Göttingen

vorgelegt von
Wafi Abed Zaidan Mohammad Dahman
aus Gaza (Barbara) - Palästina

Göttingen 2010

D7

Referent: Prof. Dr. Jens Grabowski

Koreferenten: Prof. Dr. Dieter Hogrefe, Prof. Dr. Helmut Neukirchen

Tag der mündlichen Prüfung: 05 Juli 2010

Abstract

Web services are currently one of the most important technologies for enabling an effective communication between and within distributed systems. The Web services technology relies on widely used and well-adopted technologies and open standards. A current trend in software engineering is model-based software development. One of the main goals of model-based software development is the efficient production of high quality software.

This thesis presents a comprehensive approach for the model-based development of Web services. The approach is based on a Web service profile for the Unified Modeling Language (UML), which allows an efficient definition of complete Web service models. Such Web service models allow the generation of the complete source code and the corresponding platform-specific configuration files necessary in order to run the modelled Web services. The code generation is realised by means of transformation and code generation rules defined in the Xpand transformation language. In addition to the UML Web service profile and the Xpand transformation rules, a straightforward development model for the application of the profile is proposed. The feasibility of the proposed approach for the model-based development of Web services is validated by implementing a library system Web service.

Zusammenfassung

Web Services sind gegenwärtig eine der wichtigsten Technologien, die eine effektive Kommunikation in verteilten Systemen ermöglicht. Die Web Services Technologie basiert auf weitverbreiteten und allgemein akzeptierten Technologien und offenen Standards. Ein aktueller Trend in der Softwaretechnik ist die modellbasierte Softwareentwicklung. Die effiziente Produktion von qualitativ hochwertiger Software ist eines der Hauptziele der modellbasierten Softwareentwicklung.

In dieser Arbeit wird ein umfassendes Verfahren für die modellbasierte Entwicklung von Web Services vorgestellt. Das Verfahren basiert auf der Definition eines Web Service Profils für die Unified Modeling Language (UML), das eine effiziente Definition von Modellen für Web Services ermöglicht. Zur Ausführung eines Web Services wird aus dem zugehörigen Web Service Modell der komplette Quellcode inklusive der zugehörigen Konfigurationsdateien generiert. Die Regeln zur Transformation und Quelltextgenerierung werden mit der Xpand Sprache definiert. Neben dem erwähnten UML Profil für Web Services und den Xpand-Regeln wird eine einfache Vorgehensweise zur Definition von Web Services mit Hilfe des UML Profils vorgeschlagen. Die Anwendbarkeit des in dieser Doktorarbeit entwickelten Verfahrens wird durch die Implementierung eines Web Service-basierten Bibliothekssystems nachgewiesen.

Contents

Contents	i
1 Introduction	1
1.1 Problem Statement	2
1.2 Thesis Contribution	2
1.2.1 UML Profile for Web Services	3
1.2.2 Web Services Development Model	3
1.2.3 Case Study	4
1.3 Related Work	4
1.4 Structure of the Thesis	6
2 Foundations	9
2.1 Service Oriented Architecture and Web Services	9
2.1.1 Advantages of Web Services	10
2.1.2 Web Services Standards	11
2.2 Unified Modeling Language	12
2.2.1 The Evolution of UML	12
2.2.2 UML Diagrams	12
2.2.3 UML Metamodelling	16
2.2.4 Unified Modeling Language Extension Mechanism	16
2.3 Model Driven Architecture	19
2.3.1 Computation Independent Model	20
2.3.2 Platform Independent Model	21
2.3.3 Platform Model	21
2.3.4 Platform Specific Model	21
2.4 Model Transformation and Code Generation	21
2.4.1 Code Generation	22
2.4.2 Rules for Code Generation	22
2.4.3 Integrating Manual Code	23
2.4.4 Benefits of Code Generation	24

2.4.5	Code Generation in Example	25
3	UML Profile for Web Services	33
3.1	Web Services Basic Extensions	34
3.1.1	WebService	36
3.1.2	DataContainer	37
3.1.3	DataElement	37
3.1.4	ProxyImplementation	38
3.1.5	ProxyMethod	38
3.1.6	Client	39
3.1.7	ClientMain	40
3.2	Making Web Services Executable	40
3.2.1	Executable State Machines	41
3.2.2	Auxiliary Extensions	47
3.3	Profile Implementation	49
3.3.1	Model Transformation and Code Generation	49
3.3.2	Implementation Environment	62
3.4	Summary	68
4	Web Services Development Model	69
4.1	Requirements Analysis	70
4.1.1	Requirements Elicitation	70
4.1.2	Requirements Specification and Modelling	70
4.2	Web Service Design	73
4.2.1	Realisation of Architecture Design	73
4.2.2	Realisation of Behaviour Design	76
4.2.3	Designing the Web Services Platform	77
4.3	Web Service Implementation	78
4.4	Summary	79
5	Case Study: Library System Web Service	81
5.1	Service Description	81
5.2	Library System Web Service Analysis	82
5.3	Library System Web Service Design	85
5.3.1	Identifying and Allocating UML Extensions for the Library System Web Service	85
5.3.2	Refinement of Library System Web Service Architecture	86
5.3.3	Representing Library System Web Service Behaviour	87
5.4	Library System Web Service Implementation	88
5.4.1	Running the Generator	89
5.4.2	Executing the Library System Web Service	96
5.5	Summary	96

6 Conclusion	99
6.1 Summary	99
6.2 Discussion	100
6.3 Outlook	101
Bibliography	103
A Case Study Model	109
B Transformation Rules for Generating Java Code	113
B.1 CommonTemplate Template File	113
B.2 DataContainer Template File	116
B.3 WebService Template File	118
B.4 Proxy Template File	119
B.5 Client Template file	120
B.6 MyExtensions Xtend File	122
C Generated Java Source Code	125
C.1 Java Classes in the Service Provider Side	125
C.1.1 Java Classes in <i>data</i> Folder	125
C.1.2 Java Classes in <i>service</i> Folder	127
C.2 Java Classes in the Service Client Side	129
C.2.1 Java Classes in <i>client</i> Folder	129
D Transformation Rules for Generating Configuration Files	133
D.1 XmlFiles Template File	133
D.2 build.xml File	135
D.3 services.xml File	137
E Transformation Rules for Generating the README File	139
E.1 README Template File	139
E.2 README.txt File	140
F Executing the Web Service	143
F.1 Web Service Engine	143
F.1.1 Apache Axis2/Java	143
F.1.2 Apache Axis2/C	143
F.2 Setting the Environments	144
F.2.1 Web Service Engine	144
F.2.2 Application Server	144
F.2.3 Operating System	144
F.2.4 Additional Considerations	144
List of Symbols and Abbreviations	147

List of Figures	149
Listings	151
List of Tables	153

Acknowledgements

With great reverence, I must thank ALLAH the creator of the universe for giving me the patience and ability to complete this part of my academic life.

I wish to thank all those who helped me in some way during this thesis, without whom I could not have completed this project.

I would like to express my sincere appreciation to my supervisor Prof. Dr. Jens Grabowski, and co-supervisors Prof. Dr. Helmut Neukirchen, and Prof. Dr. Dieter Hogrefe for their guidance, continuous encouragement, and support throughout my study and during the preparation of this thesis. Thanks are extended to Prof. Dr. Jens Grabowski for the nice working atmosphere. I cannot forget Mrs. Annette Kadziora, and Mr. Gunnar Krull.

I am grateful to Mr. Ulrich Brawand for his support and the valuable time we spent together in discussing different technical issues in the research, and to Karsten Thoms, and Christian Dietrich for their valuable help via the forums of OpenArchitectureWare and Eclipse.

I would like to acknowledge and extend my heartfelt gratitude to the kind persons who made the completion of this thesis possible. I appreciate highly the efforts of Akhtar Ali Jalbani, Benjamin Zeiss, Edith Werner, Dr. Hanan Almansi, Dr. Nizar Aouni, Philip Makedoniski, Dr. Saad Suleiman, and Thomas Rings in the proof-reading.

I am grateful the German Academic Exchange Service (DAAD) for the financial support. Great appreciation goes to the contact persons, Ms. Cornelia Hanzlik-Rudolph, Mrs. Kirsten Bönninghausen, Ms. Andrea Gerecke, and Ms. Karla Barth. Many thanks to the DAAD members in Jerusalem in Palestine, Dr. Helga Baumgarten, and Mrs. Eveline Muhareb.

Last but not least, huge thanks to my mother, my father, beloved Dalia and Zaidan. The love, encouragement, and support from them have been outstanding. Words just can not express how grateful I am for that.

Chapter 1

Introduction

Web services are emerging day by day, and gaining more involvement in businesses especially over the Web. The development of Web services has become one of the hot topics, which deserve more emphasis and investigation. Web services vary in size and complexity from a single and small Web service that performs simple calculations to large-scale Web services that manage multi-nation enterprises and serve multiple businesses in different locations in the world. They enable the interaction between various distributed systems over the Web. The interaction in Web services is normally done between a client agent that represents the application requesting the Web service, and a provider agent that represents the application offering the Web service and providing the responses to the clients. Web services are a key tool for businesses to promote their existence from narrow markets to wider horizons, where they can enlarge their profits by reaching a greater number of customers with minimum costs.

Similar to other software applications, Web services have a specific structure and behaviour. The structure is the static part of Web services, which is composed of the candidate objects and entities and their associations. The behaviour is the dynamic part, which represents how the Web service behaves in terms of sending requests, preparing responses to these requests, and how they will be sent back to the clients. Both parts are essential in the development of Web services in this thesis. Web services use Xtensible Mark-up Language (XML) [XML], which is the key for enabling the interoperability. Interoperability means that different types of systems running on a large diversity of platforms can communicate and exchange data without any troubles.

The Unified Modeling Language (UML) [UML10a, UML10b, UMLa] gains great acceptance among software developers, not only because of its standardisation

by the Object Management Group (OMG) [OMG], but also because of the high support from tool vendors, and the increasing availability of open source tools. The tools make the development of software with UML easier. Some of these tools also consider quality attributes (e.g. consistency) of the modelling and the software development processes and enable the generation of source code out of UML models. UML offers an extension mechanism, where UML metamodels can be extended to fit the requirements of specific domains (e.g. Web services).

The Model Driven Architecture (MDA) [MDA03] is a framework for the development of software systems based on different types of models that vary in the level of abstraction. MDA recommends UML as a modelling notation for the generation of executable systems. This research will investigate this issue, and check whether it is possible to generate executable Web services from UML models following the principles of MDA.

The methodology in the field of systems development refers to a comprehensive framework that includes all activities that follow a specific life cycle in order to develop systems according to well-defined and complete specifications. Depending on the life cycle of the methodology, different activities may be found, such as planning, requirements analysis and design, implementation, testing, and maintenance. The activities can be performed in several fashions, e.g. sequential, parallel, iterative, or incremental. Systems developers have been using different types of methodologies for systems development. The Rational Unified Process (RUP) [JBR03] and the V-Modell XT [VMX06] are examples of the methodologies that may be used in systems development. The thesis will define a development model for Web services based on the best practises of the RUP. Software development methodologies can cover several fields like software, hardware, projects management, finance, and even social aspects in order to produce systems of high quality.

1.1 Problem Statement

The problem examined in this thesis can be summarised in the following hypothesis:

It is possible to develop executable Web services from UML models. The model transformation and the code generation techniques are good enough to produce executable Web services.

1.2 Thesis Contribution

In order to resolve the problem statement stated in the previous section and to provide a reliable solution, the following contributions are made.

1.2.1 UML Profile for Web Services

UML profiles are used to customise UML to fit specific application domains. A set of UML extensions has been defined to represent the static structure and dynamic behaviour of the Web services. This thesis defines a UML Profile for Web Services (UP4WS), which will be used in the model transformation and code generation process.

1.2.1.1 UP4WS Extensions

The extensions defined in the UML Profile for Web Services (UP4WS) have two purposes. The first is to represent the basic and mandatory extensions needed for any Web services application, while the second is to make UML executable and to enable the generation of source code and output files from the UML models. The UP4WS has been defined based on thorough studies of different kinds of Web services implementation, and how they are deployed. Class diagrams are used to represent the static structure of Web services, while state machine diagrams are used to represent the dynamic behaviour of the Web services.

1.2.1.2 UP4WS Implementation

The main activity is to transform the model and generate the output files in order to execute the Web service. This thesis presents a straightforward method for the UP4WS implementation including the definition of the transformation rules for the code generation process and environment configurations. The transformation rules are defined in Xpand [XPA]. In order to execute the code generation process, the generator has to be configured. The same is also applicable to the execution of the Web service, where the platform specifications have to be established and configured correctly. All of this is described in the profile implementation and the configuration instructions.

1.2.2 Web Services Development Model

The Web Services Development Model (WSDM) specifies a set of tasks to develop Web services. The main target of the WSDM is to define a straightforward model for Web services development. The WSDM constrains the use of the UP4WS and its extensions. The tasks of the WSDM are developed to be synchronised with the profile definition and implementation as well. The WSDM is composed of three main tasks described below.

1.2.2.1 Web Service Requirements Analysis

The Web service requirements analysis task is dedicated to capture and gather the requirements for the target Web service. This includes the identification of

the exact services that shall be provided. For this task, the **WSDM** proposes two types of **UML** diagrams, i.e. use case diagrams, and class diagrams.

1.2.2.2 Web Service Design

This Web service design task realises the requirements captured in the requirements analysis by extending and refining them. The **UML** extensions defined by the **UP4WS** are identified and allocated to the model elements. At this task, class and state machine diagrams shall be used. From the class diagrams, the structure of the source code can be generated, while the state machine diagrams enable the generation of the behaviour of the Web service.

1.2.2.3 Web Service Implementation

The Web service implementation task represents a realisation of the **UP4WS** implementation. This includes the definition of the transformation rules, and setting the environments for the modelling, code generation, and executing the Web service.

1.2.3 Case Study

UP4WS and **WSDM** are validated by implementing a library system Web service as a case study.

1.3 Related Work

The work presented in this thesis spans a variety of techniques that together achieve the main goal of this thesis. It includes the definition of **UML** profiles for specific domains (i.e. Web services). In addition, it defines an approach for model transformation and code generation. The examined work covers different perspectives that relate to the goal of this thesis. Each perspective corresponds to a sub-goal of this thesis. For example, some of the examined work tries to model Web services with **UML**. However, their view to Web services is only constrained to Web Service Description Language (**WSDL**) documents [SCV03, GSSO04, Arm02]. Others take the composition of Web services into consideration [SGS04, TDE03]. Another category of the examined work concentrates on the generation of source code out of **UML** models independent of the type of the application and the implementation platform [UN09].

Marcos et al. [SCV03] present a **UML** extension to model **WSDL** as interfaces for Web services. They describe a **UML** metamodel for **WSDL** representing all possible extensions for the concrete and abstract elements of **WSDL**. They target the generation of **WSDL** descriptions from **UML** models. Armstrong [Arm02]

describes the modelling of Web services with **UML** presenting a general model of Web services standards (i.e. Web Service Description Language (**WSDL**), Simple Object Access Protocol (**SOAP**), and Universal, Description, Discovery, and Integration (**UDDI**)) and their associations in **UML**.

Skogan et al. [SGS04] introduce a **UML**-based model-driven method for Web service composition. This method shows the possibility of using model transformation to get executable models from composite Web services models. In addition, they present a **UML** profile and guidelines for modelling composite Web services. Thöne et al. [TDE03] define a similar method to describe a **UML**-Web Service Composition (**WSC**) profile as a replacement for the Business Process Execution Language for Web Services (**BPEL4WS**), which is a language for the formal specification of business processes and business interaction protocols [BPE03].

The conversion rules between **UML** and Web services described by **WSDL** documents and **XML** Schema is provided by Gronme et al. in [GSS04]. They recommend the modelling of Web services by **UML** disregarding the **WSDL** constructs and present a mapping between **WSDL**-independent **UML** models and the service description in **WSDL**. They find out that: 1) A **WSDL**-independent **UML** model of a web service is better than a **WSDL**-dependent model or pure **WSDL** in explaining the service, 2) **WSDL**-independent **UML** models make the building of Web services simpler, especially if the Web service is complex, and 3) reverse and forward engineering between **WSDL** specifications and **WSDL**-independent **UML** models is possible for all kinds of services.

Usman and Nadeem [UN09] develop a code generation tool known as UJECTOR. The UJECTOR tool generates code from **UML** class, sequence, and activity diagrams. The structure of the code is generated from the class diagrams, while the sequence diagrams with incorporation of the activity diagrams generate the code for the methods in the classes. However, they did not specify Web services as a target application for the code generation process.

A quick review for the related work shows that the work performed does not satisfy the main goal of this thesis, which is the development of executable Web services based on **UML** models. The related work sees Web services only from one angle, which is the **WSDL** documents. **WSDL** represents only the Web service interface, which describes the Web service and how it can be communicated. Examined researches define **UML** metamodels for **WSDL** as a Web service, or for the composition of Web services. They do not take the execution of Web services into consideration, and thus, not able to generate the complete source code of the implementation. The researches that concentrate on the generation

of source code from **UML** do not consider any application domain (e.g. Web services). They try to define a mapping between **UML** models and source code constructs to generate an executable source code. However, the source code does not contain any platform specifications, since the original model is not defined for a specific domain. The goal of the thesis is to generate a complete source code for Web services that can be executed on a particular platform. This includes the generation of the configuration files that enable the source code to run on the selected platform.

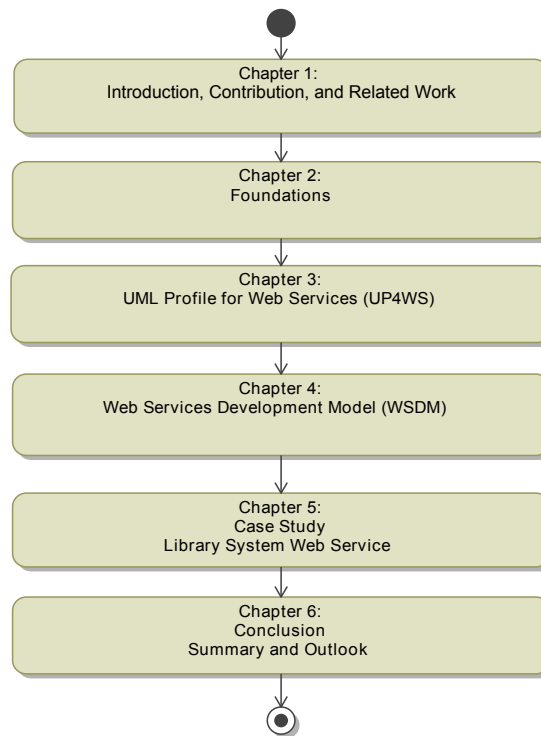


Figure 1.1: Thesis Structure Overview

1.4 Structure of the Thesis

The thesis is structured in the following order which is depicted in Figure 1.1. This chapter introduces the problem statement of the thesis, its contributions, and the related work. The foundations are presented in the second chapter. The third chapter explains the UML Profile for Web Services (**UP4WS**) and the extension mechanism followed to define the characteristics of the Web services. In addition, it describes the **UP4WS** implementation. Chapter four presents a description of a Web Services Development Model (**WSDM**), and shows how **UML** can be integrated into it. The **UP4WS** and the **WSDM** are validated in a case

study in chapter five. The last chapter concludes the thesis by summarising it and presenting an outlook on the future work. The appendices are presented at the end of the thesis.

Chapter 2

Foundations

This chapter introduces the underlying technologies that are related to the main goal of the thesis by presenting a description of each technology discussed within the thesis. Firstly, it presents an overview of Service Oriented Architecture (SOA) and Web services. Then, it explains the Unified Modeling Language (UML) as a modelling notation that is used within the thesis. Since the thesis follows the Model Driven Architecture (MDA) approach, an overview of it, and its main models are discussed. In addition, the chapter explains the model transformation and code generation approach supported with an example.

2.1 Service Oriented Architecture and Web Services

The Service Oriented Architecture (SOA) [Erl04] describes the communication between two or more services over the Web or any possible network. The purpose of the communication is to perform some functionality that can be very simple or very complex. SOA has several methods of implementations, such as the Common Object Request Broker Architecture (CORBA) [COR02], and Web services. Web services are the most popular implementation of the SOA. Therefore, they will be described in the following as a real example of the SOA. The World Wide Web Consortium (W3C) [W3C07] defines Web services as follows: *"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards"*. The word *Web* in the name of Web services does not mean that it is a Web application, rather it relies on Web technologies like Hyper Text Transfer Protocol (HTTP) [Con03]. The main idea and target of Web services

is to enhance interoperability of distributed system over networks especially the Internet, mainly clients and servers, where both sides exchange XML messages. Web services can interact with and invoke each other, and be aggregated to form larger Web services with additional functions. The International Business Machines (IBM) company has defined a model to describe interactions in any possible implementation of SOA as shown in Figure 2.1 [DDDT03]. The model shows three main roles involved in the architecture, Deitel et al. [DDDT03] define these roles as follows:

1. Service Provider is *"a server or system that makes a Web service available over a network, such as the Internet"*.
2. Service Requester is *"a networked server or system that accesses and employs a Web service and interacts with a service broker to find a Web service that fills a specific computing needs"*.
3. Service Broker is *"a networked server or system that maintains a directory or clearinghouse for Web services"*.

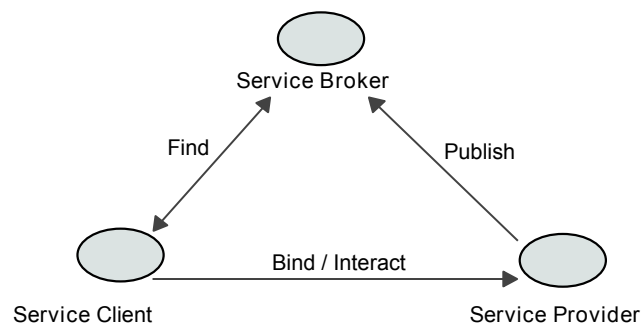


Figure 2.1: Roles in the Service Oriented Architecture (SOA)

2.1.1 Advantages of Web Services

Before and in parallel with the emergence of Web services, similar technologies have appeared (e.g. CORBA [COR02]). However, Web services have advantages compared to other technologies. Deitel et al. [DDDT03] identified the following set of Web services advantages:

- Web services rely on open and textual standards that enable the communication between different platforms and languages.
- Web services rely on existing infrastructure and software. Therefore, it is easy to implement them without new investments in a new infrastructure or software.

- It is possible to implement Web services in an incremental manner. This will reduce the costs of adopting Web services and switching to the related technologies.
- The most important advantage of Web services is actually the interoperability, since it relies on open standards, which enable the smooth communication between different platforms and applications.

2.1.2 Web Services Standards

Web services standards seem to be promising and problematic at the same time. They are promising because they depend basically on **XML** and **HTTP**. **XML** can operate on any platform and **HTTP** can be sent through firewalls and proxies without problems. The problematic side in Web services standards is their variety and massiveness. There is a large number of Web services standards, which are published by different parties and institutes that have different levels of influence on Web services. The large number of Web services standards can make the adoption and publishing of Web services difficult due to the conflicts that may arise among these standards. An overview of Web services standards are found in [WSS07, Til07]. The most essential and popular Web service standards are the Web Service Description Language (**WSDL**) [WSD07], the Simple Object Access Protocol (**SOAP**) [SOA07], and the Universal, Description, Discovery, and Integration (**UDDI**) [UDD04].

2.1.2.1 Web Service Description Language

Deitel et al. [DDDT03] define the Web Service Description Language (**WSDL**) as follows: *"The XML-based language, through which Web services describe themselves to developers and applications over the Internet. WSDL descriptions convey the methods that a Web service provides and how those methods can be accessed"*. Since **WSDL** is based on **XML**, this means that it is interoperable and can be understood similarly by different platforms. The **WSDL** document defines some aspects concerning the service, such as functionalities, location on the network, and how it can be accessed. **WSDL** plays an important role within Web services, since it defines how to invoke the service and the expected response from it [Con03]. The **WSDL** is posted by the service provider to an **XML** registry of **WSDL** repository, where clients look for the relevant services. For any type of Web services interaction, all parties need to have access to the same **WSDL** document. This enables the same interpretation of messages and guarantees that each party handles the message in the correct way. Additional advantage of using **WSDL** is that it makes any implementation of any application possible, since it provides a common format for encoding and decoding of messages with respect to any application running in the backend [New02].

2.2 Unified Modeling Language

The Object Management Group (OMG) [OMG] defines UML as follows: *“The Unified Modeling Language (UML) is a graphical language for visualising, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system’s blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schema, and reusable software components”*. According to [AN05, Fow04], UML is a general purpose graphical modelling language for the development of systems. UML is the leading the modelling language in the field of systems engineering and has gained much tool support. The tool support for UML influences positively its usability and readability, which enables the effective building of systems. Despite the fact that UML is mostly used with object oriented systems, it has the capability to develop other types of systems through its flexible extension mechanism or profiles.

2.2.1 The Evolution of UML

The history of UML started few years before 1994 when different modelling languages concerned with object oriented development existed. Each one had its own strengths and weaknesses. The most popular techniques were the Booch Methodology [BOM], the Object Modeling Technique (OMT) [OMT], and the Rational Objectory Methodology [ROM]. In 1994, the first attempt to unify the languages and techniques took place. In 1996, OMG issued a Request For Proposal (RFP) for a graphical modelling language. In response, OMG received UML, which became an OMG standard in 1997. In 2000, UML versions 1.x were released. These versions offered additional features, where action semantics could be used to enable the execution of UML. The term is known as Executable UML (xUML) [UMLb]. In 2004, UML versions 2.x became available. These versions added new visual syntax either instead of the 1.x versions or new in addition to them without changing their basic principles [AN05].

2.2.2 UML Diagrams

Unified Modeling Language v.2.x (UML2) specifies fourteen types of diagrams to enable the modelling of both static structure and dynamic behaviour of systems, and help in managing the entire development process. UML2 specifications classify the diagrams into two categories, i.e. structural and behavioural diagrams. The structural diagrams are the ones that define the static view of the system. They represent the entities involved in the system and the relationship among them. Structural diagrams include: package, class, component, object, composite structure, deployment, and profile diagrams.

The behavioural diagrams represent the dynamic view of the system, and how the entities behave and communicate in order to produce the desired behaviour of the system under development. Behavioural diagrams include: use case, activity, state machine, and interaction diagrams, which include sequence, interaction overview, communication, and timing diagrams [AN05, UML10b].

Taking the decision to adopt UML in the development process does not mean that all UML diagrams must be used. It is up to the modeller to decide, which diagrams to use in the development process. The decision is usually based on the nature of the system, the domain, the functionalities, as well as the preferences of the modeller, since some UML diagrams might in some cases replace each other. In this thesis, three types of UML diagrams are used, i.e. a) the use case diagrams to capture the requirements of the Web service application and to represent the exact services, which are provided by the Web service, b) the class diagrams to build the Web service architecture and identify its main entities and their associations and responsibilities, and c) the state machine diagrams to model the behaviour of the Web service and its single objects. In the following, a description of a specific set of the UML diagrams that are used in this thesis is presented.

2.2.2.1 Use Case Diagrams

A use case diagram is *"a diagram that shows a set of use cases and actors and their relationships; use case diagrams address the static use-case view of a system"* [JBR03]. Use case diagrams are composed of three main elements, which have to be identified before creating them.

- **System Boundary:** represents the boundaries that distinguish the system from the rest of the world. It shows the internal parts of the system, which appear inside and outside the boundaries.
- **Actors:** are located outside the system boundary and communicate directly with the system by either sending or receiving data to/from the system, or both. Identifying the actors is very important to develop the system effectively. In order to identify them, it is necessary to specify who interacts with the system, what uses the system, and what is the system using? [AN05].
- **Use cases:** are defined in [JBR04] as *"a specification of sequences or actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside actors"*. A use case represents a specific function that the system should do. The use case is initiated by the actor and written from the actor's point of view. Identification of the use cases is also important and they can be recognised by

deciding what the system is supposed to do for each actor, and the way, in which each actor will use the system.

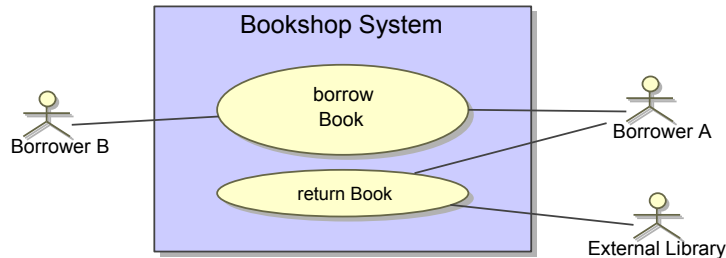


Figure 2.2: Sample Use Case Diagram

Figure 2.2 presents a sample use case diagram that shows the main elements in any use case diagram. The diagram shows a *Bookshop System* that offers two functionalities represented as use cases *borrow book* and *return book*. The use cases are associated with three actors that exchange communication with the system, i.e. *Borrower A*, *Borrower B*, and *External Library*; the third actor represents an external system. The use cases are surrounded with a rectangle that specifies the boundaries of the *Bookshop System*.

2.2.2.2 State Machine Diagrams

A state machine diagram is "a diagram that shows a state machine, with emphasis on the flow of control between states" [Sco04]. State machine diagrams are used heavily in modelling the dynamic behaviour of the system. They aim at representing the behaviour of a single entity or object within the system. States, events, and transitions are the main elements that can form any state machine diagram. A state is "a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event" [JBR04]. An event is the specification of a noteworthy occurrence that has location in time and space [JBR04]. A transition is "the movement from one state to another as a result of an event occurrence" [AN05]. Each state machine should have a starting state (filled circle), from which the transition(s) begin(s), and may also have a final state (bull's eye). UML2 specifies two types of state machine diagrams, i.e. behaviour state machines, and protocol state machines. Behaviour state machine specifies the behaviour of a classifier, while the protocol state machine specifies a protocol of a classifier via conditions, results, and ordering of operation calls. Modellers do not often differentiate between the two types although protocol state machines come with the keyword *{protocol}* after the name of the state machine. Furthermore, protocol state machines can not specify actions, which can only be specified by behaviour state machines. [AN05]

Figure 2.3 shows a sample state machine diagram for a *Book* object. The state machine diagram contains two states, i.e. *Available* and *Borrowed* that can be triggered by the operations, i.e. *returnBook()* and *borrowBook()* respectively to change the state of the *Book* object.

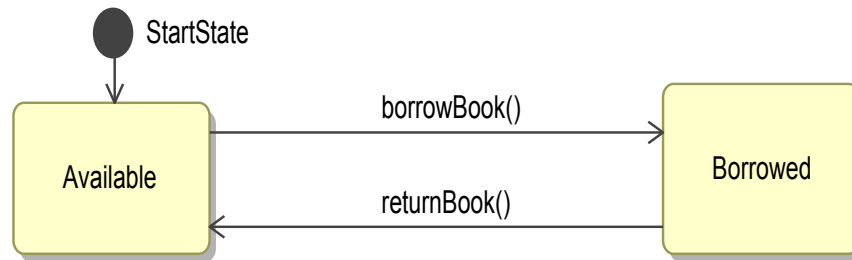


Figure 2.3: Sample State Machine Diagram

2.2.2.3 Class Diagrams

A class diagram is "a diagram that shows a set of classes, interfaces, and collaborations and their relationships; class diagrams address the static design view of a system; a diagram that shows a collection of declarative (static) behaviour" [JBR03]. Jacobson et al. [JBR04] define the *Class* as "The descriptor of a set of objects that share the same attributes, operations, methods, relationships, and behaviour". A class is seen as a container of objects that must have the same operations, attributes, and associations of that class, but with different attribute values. Class diagrams are used in any system development process that uses UML as a modelling notation. They are helpful in the development process from the very beginning, where they can be used for identifying system requirements and its entities. For example, they construct the initial system architecture in the analysis phase, while in the design phase, they are refined and extended to represent the complete system specifications. Any class diagram is mainly composed of classes and associations between them. There are many types of associations in the class diagram, e.g. inheritance, aggregation, composition, and dependency, which can be used for several purposes. Figure 2.4 shows a sample class diagram, which is composed of three classes, i.e. *Book*, *Publisher*, and *Borrower*. The diagram shows the attributes and operations inside those classes. In addition, the diagram shows different elements, such as associations (i.e. *Book-Publisher* and *Book-Borrower*), role names (i.e. *publisher*, *borrower*), and multiplicities (i.e. *1*, *1..0*, *1..**).

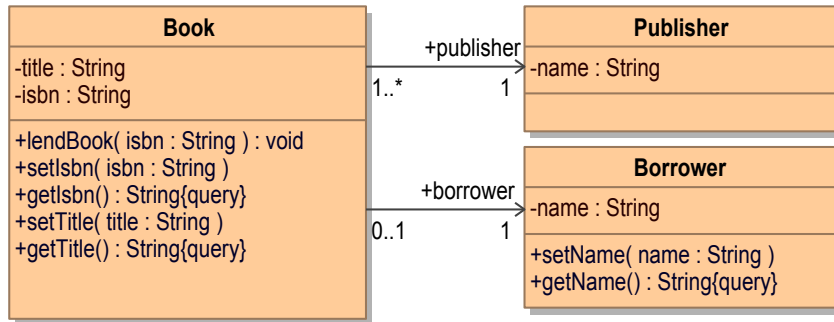


Figure 2.4: Sample Class Diagram

2.2.3 UML Metamodelling

The **OMG** modelling architecture for **UML** is composed of four layers, which describe different conceptual levels of abstraction. The layers are referred to as M0, M1, M2, and M3.

- M0 represents user instances or objects at runtime, e.g. Publisher: PublisherA
- M1 represents snapshot of the user model, e.g. classes, and associations.
- M2 represents the metamodel level, e.g. **UML** and Common Warehouse Model (**CWM**). This level defines a language for specifying the models.
- M3 forms the foundation of the metamodelling hierarchy. This layer is responsible for specifying the metamodels.

Figure 2.5 shows these layers together with an example. This example illustrates the four layers with sample elements. Layer M0 contains the user instances, while layer M1 involves the corresponding objects for those instances. Layer M2 contains the metamodelling elements, where **UML** and **CWM** reside. Both **UML** and **CWM** are instances of the Meta Object Facility (**MOF**), which reside in the M3 layer.

2.2.4 Unified Modeling Language Extension Mechanism

UML is a general purpose modelling language and not dedicated to a specific type of systems or domains. Moreover, further specialisations and extensions to allow domain-specific models could be valuable. The **OMG** has standardised many **UML** profiles to fit different types of systems and technologies, such as **UML** Profile for **CORBA** [COR02]. The **OMG** defines two approaches for defining a domain-specific models, Fuentes and Vallecillo [FV04] described them as follows:

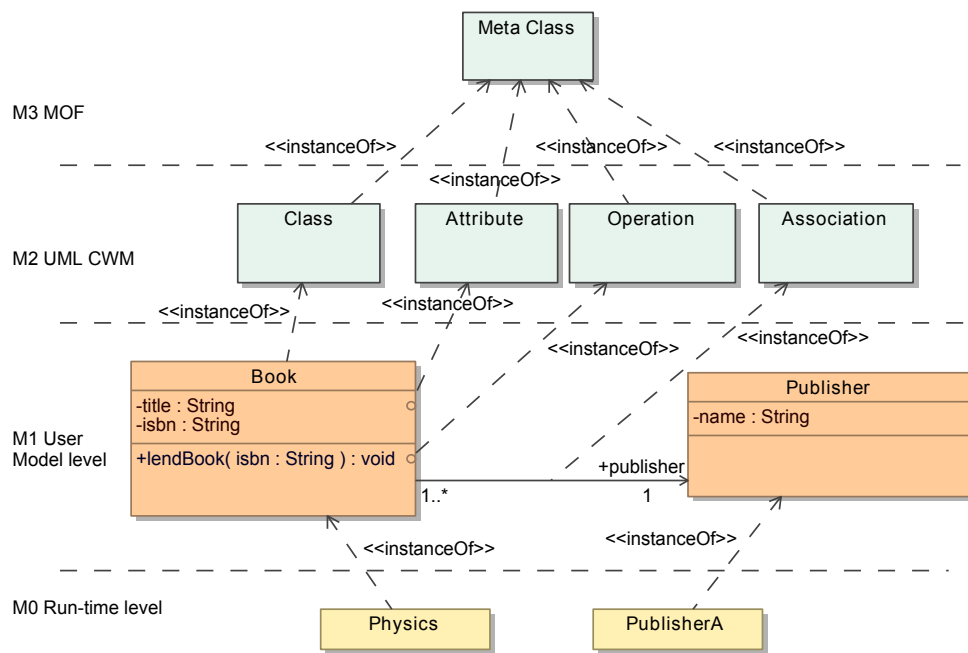


Figure 2.5: UML Metamodel Layers (Example)

- Defining a new language that is independent of UML and based on the specifications of the OMG. The new language must comply with the OMG standards, but has nothing to do the UML. It has its own syntax and semantics. An example for this approach is the definition of CWM [CWM03], which resides in the M2 layer (Figure 2.5).
- Defining a new UML profile to fit the domain-specific characteristics. The new profile must commit to the UML metamodel and impose no modifications on its semantics or syntax. For this purpose, UML offers three types of extension mechanisms, i.e. *stereotypes*, *constraints*, and *tagged values*. The extension mechanism can be used to enable the adaptation of UML to fit a specific application domain without changing or modifying the UML metamodel. For example, in a Java profile, the generalisation between classes is restricted to single inheritance, which is not the case in UML. Therefore, additional constraints can be added to restrict the inheritance in UML.

2.2.4.1 Defining UML Profiles

UML offers an extension mechanism, where new UML profile can be defined to adapt UML for specific domains. The extension mechanism is usually defined in terms of UML profiles. Arlow and Neustadt [AN05] define a UML profile as "a collection of *stereotypes*, *tagged values*, and *constraints*".

- **Stereotypes:** are defined in the **UML2** specifications as follows: “a stereotype defines how an existing metaclass may be extended, and enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass” [UML10b]. Jacobson [JBR04] define stereotypes as “a variation of an existing model element with the same form (such as attributes and relationships) but with modified intent”. Via stereotypes, it is possible to define new elements dependent on **UML** metaclasses. This is done by putting the name of the stereotype between guillemets (e.g. «MyFirstStereotype»). Constraints and tagged values can be attached to the stereotypes. Images and colours are also possible although the latter is not recommended, since it might lead to misinterpretation of the model elements [Fow04].
- **Tagged Values:** are properties attached to the modelling elements with value for each. They are normally associated with stereotypes and applied by the model elements extending those stereotypes. They follow a simple syntax, i.e. *myFirstTag = myFirstValue*, *mySecondTag = mySecondValue* and so on.
- **Constraints:** are used to restrict the use of the modelling elements for particular purpose. They are rules usually expressed in Object Constraint Language (**OCL**) [OCL06], which has a specific syntax and appear between ({..}) brackets.

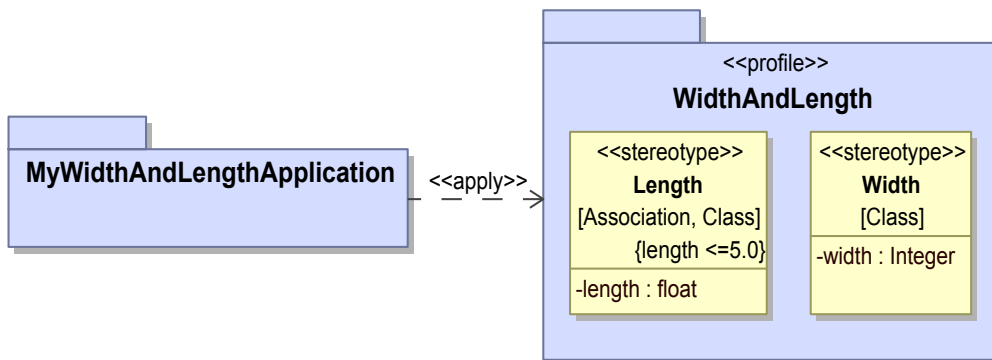


Figure 2.6: UML *WidthAndLength* Profile

Figure 2.6 illustrates a sample **UML** profile called *WidthAndLength* shown as a package with «profile» stereotype. The *WidthAndLength* profile includes two stereotypes, i.e. *Length* extending the *Association* and *Class* metaclasses, and *Width* extending the *Class* metaclass. The *Length* stereotype includes a tagged value *length* of type *float*, while the *Width* stereotype includes a tagged value *width* of type *Integer*. The *Length* stereotype contains a constraint limiting the

length to less than or equal to five ($\{\text{length} \leq 5.0\}$). The *WidthAndLength* profile is applied by *MyWidthAndLengthApplication* model.

For the definition of a **UML** profile, few important points must be taken into consideration [FV04]:

- Identify all elements, which comprise the new profile. These elements shall be represented using a **UML** metamodel.
- Once the metamodel elements are recognised, the **UML** profile is ready to be defined. This includes defining a stereotype for each needed element inside a package named «profile». Only extended elements shall be represented as stereotypes. Classes, associations, attributes, operations, states, transitions, and packages are examples of those elements.
- If any, specify tagged values for the attributes including types and initial values.
- Specify the relevant constraints on the profile elements. These constraints are usually derived from the domain.

2.3 Model Driven Architecture

Model Driven Architecture (**MDA**) is an approach for the development of software. It utilises the modelling languages as programming languages in addition to their use in the design. The main goal is to improve the quality and productivity of software development by defining models that fit in different platforms. The specifications for a certain platform can be defined at later step to form a complete model for the selected platform. In this case, the single necessity is to define a platform model for each target platform, while the original model for the software remains unchanged. **MDA** [MDA03] is an **OMG** standard, and has a specific life cycle for the development process, which starts with defining the system in an abstract way, and independent of its target platform. The second step is to specify the possible platforms for implementing the software system, then choose one of them for the execution. The final step is to transform the specifications into software, which runs on the chosen platform [MDA03]. **MDA** concentrated on the functionality and behaviour of the system and disregards the platform, which will be taken into consideration at a later phase. For this purpose, it proposes a Platform Independent Model (**PIM**), which represents the functionality and behaviour of the system without considering the implementation platform. The **PIM** will be refined into a Platform Specific Model (**PSM**), which includes the characteristics of a specific platform. Figure 2.7 represents a

UML model transformation view for MDA models, and how they relate to each other.

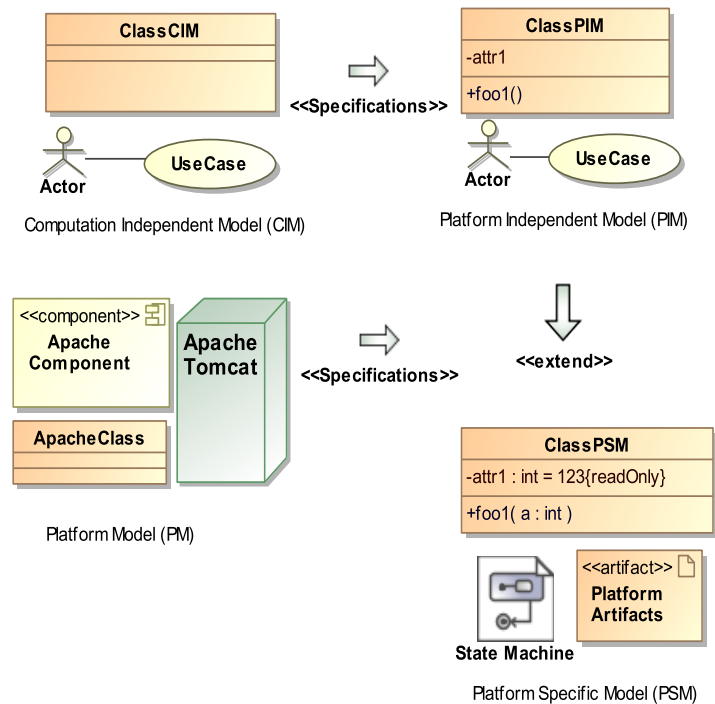


Figure 2.7: A UML View for MDA Models

MDA defines a specific set of models within its approach. Each model corresponds to a different level of abstraction. In the following a brief description of these models is presented.

2.3.1 Computation Independent Model

The Computation Independent Model (CIM) describes the system from the computation perspective, where no details of the system structure are considered. It is sometimes referred to as the domain model and targets the experts of the system domain. Therefore, it uses the terms of the domain experts. The main goal of the CIM is to fill in the gap between the domain experts who specify the system requirements and the design and technology experts who will translate these requirements into design and architecture. The CIM does not consider how the system will be implemented. It only shows its environment and what it is supposed to do, and serves as means of communication between the domain and technology experts. It should be possible to track the requirements of the CIM in both PIM and PSM [MDA03].

2.3.2 Platform Independent Model

The Platform Independent Model (**PIM**) views the system from a platform independent perspective in order to make it suitable for multiple and different platforms. It specifies the services and interfaces, which are supposed to be provided by the software without considering the target platform(s). Therefore, it contains complete specifications of the software. These specifications can be related to the enterprise, as well as the computation environment.

2.3.3 Platform Model

The Platform Model (**PM**) represents the platform, on which the software will run. It shows a set of technical parts of the target platform and services provided by it, and helps in the process of transforming the **PIM** into a **PSM**.

2.3.4 Platform Specific Model

The Platform Specific Model (**PSM**) realises and refines the **PIM** and is reached through model transformation techniques. It merges the platform independent specifications together with the details of the target platform in order to enable the generation of the code and the configuration files that make the software executable. The **PSM** should not be seen as a one-to-one mapping to the **PIM**, since the **PSM** includes more details and representations, which are not present in the **PIM** [Mil03]. Through model transformation, the code that runs on the target platform is generated from the **PSM** [MDA03]. More than one transformation may be necessary, since the abstraction gap is relatively big [Fra03].

2.4 Model Transformation and Code Generation

The model is the core for any model transformation process. A model is an abstraction of the reality and its complexity. Czarnecki and Helsen [CH06] define a model as *"an abstraction of a system or its environment, or both"*. MDA [MDA03] defines the model transformation as *"The process of converting one model to another model of the same system"*. Kleppe et al. [KWB03] present a more comprehensive definition of model transformation, which is: *"the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language"*. Although these definitions mentioned only one type of model transformation, i.e. Model-to-Model (**M2M**), this does in no way mean that other types of model transformation are not included, since a model

can be graphical or textual. Other possible types of model transformation are the Model-to-Text (M2T), Text-to-Model (T2M), and Text-to-Text (T2T). Furthermore, the model in the definition includes implicitly metamodels, in other words, it should be possible to transform a metamodel to another metamodel. This is known as a horizontal model transformation, since both the source and the target models remain at the same level. If the transformation moves the model from one level to another, it is known as a vertical model transformation [MG06]. Another distinction in model transformation is between endogenous (or rephrasing) and exogenous transformation. The former corresponds to the transformation conformity to a single metamodel, while the latter corresponds to the transformation procedures on several metamodels [CH06]. Mens and Van Gorp [MG06] proposed additional feature for model transformation, where they emphasise that model transformation should be possible with multiple source models and/or multiple target models. For example, in terms of Web services, it could be necessary to define a source model for the target Web service, and another one for the platform, where the Web service will be deployed and implemented. Both will be combined or merged to generate the target source code that runs on the specified platform.

This thesis focuses on the M2T transformation, since it targets the generation of executable source code for Web services from UML models. Code generation is explained in more detail in Section 2.4.1.

2.4.1 Code Generation

Code generation is the other common name of the Model-to-Text (M2T) or Model-to-Code (M2C) transformation, where textual code is generated from a given model or some specifications. Code generation is *writing a program that writes another program* [Her03], or *code that writes code* [Dol04], and is known as *metaprograms*, which receive some specifications or model as input and generate the source code as output [SV06]. The generated code is supposed to be ready for interpretation or compilation. Therefore, it should be complete and executable from the modeller's point of view [KT08].

2.4.2 Rules for Code Generation

Herrington [Her03] specifies the following set of rules to follow when generating code:

- Understand the framework, write the structure of the code manually, and reuse it in the generator templates and the definition of the transformation rules.

- Respect manual coding, since it is sometimes not avoidable. Make its part as small as possible, because the time of development is valuable. Repetitive parts of the code must not be written manually.
- Document the process by defining instructions and warnings for the users, enabling comments inside the generator and in the output as well. This will enhance the understanding of the process, and avoid re-running the generator several times without actual need.
- Make the code generation process easy, straightforward, and reusable by defining simple transformation rules for the process. Complex transformation rules will quickly become obsolete.
- Let the code generation become a coherent part of the development process. This must be considered by the developers, i.e. selection of the environment, and tools for code generation, and how this will affect the development plan.
- Beautify your generated code to follow the same coding style of the language you have chosen. This increases its readability, and help to get more acceptance among the audience.

2.4.3 Integrating Manual Code

The general and common rule in code generation is *never* modify the generated code. If the output source code is not as it should be, modify either the model or the generator. Added code to the output will be lost at each re-run of the generator. It is important to distinguish between the code written to fit some specifications (e.g. platform specifications) and that written for a very narrow purpose in the application. The former type will be applicable for all applications in the same domain and must be decided by the domain experts. The latter aims at serving some specific purposes, and is usually added by the modeller. This type can be added in one or more of the following ways proposed by [KT08].

2.4.3.1 Protected Regions

Protected regions are parts of the generator that shall not be overridden at every generator run [OAW10]. The generator shall be made aware that these parts are manually written or shall be written. To enable this feature in the generator, a specific syntax for protected regions must be inserted and the generator must know how to handle it. The Xpand transformation language (explained in Section 2.4.5) offers this capability by using the syntactical constructs illustrated in Listing 2.1 in the relevant template.

```

1 «PROTECT CSTART expression CEND expression ID expression (DISABLE)?»
2
3 // manual code can be inserted here
4
5 «ENDPROTECT»

```

Listing 2.1: Protected Regions in Xpand

Protected regions are mostly provided by **UML** tools with code generation features. The tool builds the skeleton of the code and specifies the regions, where the user can insert manual code as shown in Figure 2.8.

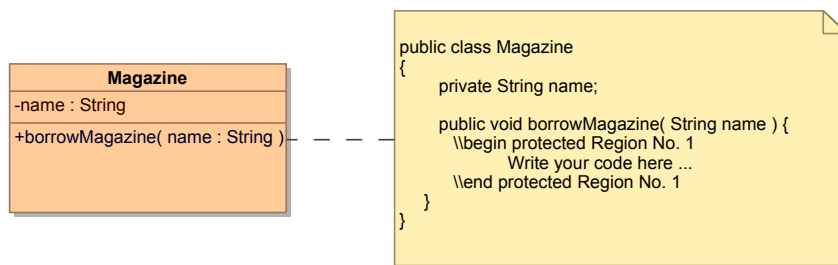


Figure 2.8: Protected Regions

The users should work on models, generators, and output when they decide to use the protected regions.

2.4.3.2 Writing Code in Models

Source code can be inserted in the source model instead of the generated source code. In this case, all the information required for code generation is located in one single source. However, the amount of the inserted code should be as small as possible in order to keep the model readable, and to ensure the efficiency of the whole code generation process.

2.4.3.3 Files Referenced by Models

Another possibility for the insertion of the manual code is to define an external and editable file, to which the model should refer. Each part of the code could reside in a separate file. It is also possible to make the generated code reference to the manual code in the file(s), which should be generated with the automated code. Referencing could be done via import statements in the generator.

2.4.4 Benefits of Code Generation

Code generation offers the developers with multiple advantages in the software development process. In the following, a summary for these advantages is presented [Her03]:

- Code generation guarantees consistency in the generated code including all entities and elements. This feature is easy to recognise in the names of the output entities. In addition, the generated code is consistent with the architecture, since it maps the architecture model. If the code is not executable or contains problems, this means that the architecture contains errors or it is irrelevant to what the system should do. Such advantages maximise the quality of the generated code.
- No massive changes are required when a change is needed, since only the template that generates the code should be changed.
- Problems are quickly identified and resolved in the template(s), where the corresponding transformation rules reside.
- Unit testing is possible, since generators are able to create unit tests for generated entities.
- It is still possible to generate code for several platforms, since the application logic differs from the implementation platform. Platform specific characteristics in the generator change, depending on the platform itself.
- Source code, configuration files, and even documentation can be generated at the same time by running the generator only once. This benefit has an influence on the productivity of the systems development, since much of the redundant work will be done automatically, and thus developers can concentrate on the most important work instead of the entire system.

2.4.5 Code Generation in Example

Xpand was initially developed by OpenArchitectureWare [OAW] as a Model-to-Text (M2T) transformation language. Since September 2009, Xpand has become part of the M2T transformation languages integrated into the eclipse platform. Xpand is a template-based transformation language for controlling the code generation process [OAW10]. It defines one or more template file(s), where each template file consists of one or more templates that are defined using the keywords *DEFINE* and *ENDDEFINE*, which is also known as *DEFINE* block, the example in Listing 2.2 shows:

```
1 «DEFINE myFirstTemplate FOR Type»  
2  
3   some statements...  
4  
5 «ENDDEFINE»
```

Listing 2.2: Template Definition

Xpand offers different commands known as *metacode* to enable accessing the metamodel or source model in order to generate the target output according to the input source model and the metamodel.

2.4.5.1 Features of Xpand

Xpand provides multiple features to enable code generation in a smooth way. The following is a set of the important features of Xpand.

2.4.5.1.1 IMPORT

The *IMPORT* enables importing namespaces and using the unqualified names contained in the imported namespace (Listing 2.3).

```
1 «IMPORT myMetamodel::myModel»
```

Listing 2.3: Import Statement

2.4.5.1.2 EXTENSION

In some cases, it is not easy to describe or specify some behavioural operations or queries in Xpand. Therefore, the Xtend language, which is heavily used in Xpand projects [XPA], can be used to define some operations, which are invoked inside Xpand templates by extending the Xtend file using the *EXTENSION* keyword (Listing 2.4).

```
1 «EXTENSION myRootFiles::myExtensionFile»
```

Listing 2.4: Extension Statement

2.4.5.1.3 FILE

The *FILE* block enables the generation of files that contain the target output, and stored in a specific location, known as *outlet*. If no outlet is specified, the file will be stored in a default source folder. The body of the file can contain any string of code or metacode (Listing 2.5).

```
1 «FILE expression [myOutlet]»
2
3   any String ...
4
5 «ENDFILE»
```

Listing 2.5: FILE Block Definition

2.4.5.1.4 FOREACH

This feature enables retrieving a specific collection and manipulating it inside the body of the *FOREACH* block (Listing 2.6).

```

1 «FOREACH expression AS variableName»
2
3   a sequence of statements using variableName to access the elements
4
5 «ENDFOREACH»

```

Listing 2.6: FOREACH Block

2.4.5.1.5 EXPAND

The *EXPAND* statement invokes another *DEFINE* block or template inside or outside the same template file. More than one *EXPAND* statement can be found in the same template (Listings 2.7 and 2.8).

```

1 «EXPAND anotherTemplate FOR this»

```

Listing 2.7: EXPAND Statement A

```

1 «EXPAND anotherTemplate FOREACH expression»

```

Listing 2.8: EXPAND Statement B

2.4.5.1.6 IF

IF statement is one of the powerful features of Xpand, since it enables applying some conditions on the output. It includes naturally the *ELSE* and *ELSEIF* statements to enable inserting nested conditions (Listing 2.9).

```

1 «IF expression»
2
3   some statements
4
5   «ELSEIF expression»
6
7     some statements
8
9   «ELSE»
10
11     some statements
12
13 «ENDIF»

```

Listing 2.9: IF Statement

2.4.5.1.7 REM

The *REM* command enables the insertion of textual comments inside the template file. The comments have no influence on the output, but they are used normally to add some explanations on the contents of the templates or the expected output (Listing 2.10).

```

1 «REM»
2
3   my comment(s)
4
5 «ENDREM»

```

Listing 2.10: REM Block

2.4.5.2 Xpand in Example

The example in Figure 2.9 shows a simple class diagram. The diagram is composed of two classes *Book* and *Publisher*, and an association between them.

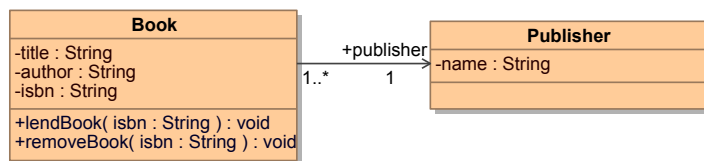


Figure 2.9: Book-Publisher Class Diagram

For the code generation of the classes in Figure 2.9, an Xpand template file as in Listing 2.11 needs to be developed. The template file contains four templates (i.e. *DEFINE* blocks):

1. **Class Template:** generates a Java class for each **UML** class. It invokes the other templates in the same file.
2. **Attribute Template:** generates the definition for each attribute in the **UML** classes.
3. **Operation Template:** generates the signature of the Java operations for each operation that appears in the **UML** class diagram.
4. **GettersAndSetters Template:** generates getter and setter operations for the private attributes in **UML** classes. The template includes an IF statement to apply this template only for attributes that have a visibility of type *private*.

The Xpand template file in Listing 2.11 generates the *Book* class in Listing 2.12, and the *Publisher* class in Listing 2.13 that both correspond to Figure 2.9.

```

1 «IMPORT uml»
2
3 «EXTENSION templates::Java»
4
5 «REM»***** Class Template *****«ENDREM»
6

```

```

7  «DEFINE classTemplate FOR uml::Class»
8    «FILE name + ".java" »
9    «visibility» class «name» {
10
11      «EXPAND attributeTemplate FOREACH ownedAttribute»
12
13      «EXPAND GettersAndSetters FOREACH ownedAttribute»
14
15      «EXPAND operationTemplate FOREACH ownedOperation»
16
17    }
18    «ENDFILE»
19 «ENDDEFINE»
20
21 «REM»***** Attribute Template *****«ENDREM»
22
23 «DEFINE attributeTemplate FOR uml::Property»
24
25   «visibility» «type.name» «name»;
26
27 «ENDDEFINE»
28
29 «REM»***** Operation Template *****«ENDREM»
30
31 «DEFINE operationTemplate FOR uml::Operation»
32   «visibility-» «IF isStatic-» static «ENDIF-»
33   «IF type==null-» void «ELSE-» «type.name»«ENDIF-» «name-»
34   («FOREACH ownedParameter.reject(e|e.direction.toString().toLowerCase()).
35   endsWith("return")» AS p SEPARATOR ", "-»
36
37   «p.type.name-» «p.name-»
38
39   «ENDFOREACH-»){
40
41     // TODO: Implement code of «name»()
42   }
43 «ENDDEFINE»
44
45 «REM»***** Getters & Setters Template *****«ENDREM»
46
47 «DEFINE GettersAndSetters FOR uml::Property»
48
49 «IF visibility.toString() == "private"»
50
51   public void set«name.toFirstUpper()»
52   («type.name» «name.toFirstLower()»){
53     this.«name» = «name.toFirstLower()»;
54   }
55
56   public «type.name» get«name.toFirstUpper()» (){
57
58     // TODO: Implement code of «class.name».«name»()
59
60     return this.«name»;
61   }
62 «ENDIF»
63 «ENDDEFINE»

```

Listing 2.11: Xpand in Example

```
1 public class Book {
2     private String title;
3     private String author;
4     private String isbn;
5     public Publisher publisher;
6
7     public void setTitle(String title) {
8         this.title = title;
9     }
10
11    public String getTitle() {
12        // TODO: Implement code of Book.title()
13        return this.title;
14    }
15
16    public void setAuthor(String author) {
17        this.author = author;
18    }
19
20    public String getAuthor() {
21        // TODO: Implement code of Book.author()
22        return this.author;
23    }
24
25    public void setIsbn(String isbn) {
26        this.isbn = isbn;
27    }
28
29    public String getIsbn() {
30        // TODO: Implement code of Book.isbn()
31        return this.isbn;
32    }
33
34    public void lendBook(String isbn) {
35
36        // TODO: Implement code of lendBook()
37    }
38
39    public void removeBook(String isbn) {
40
41        // TODO: Implement code of removeBook()
42    }
43 }
```

Listing 2.12: Java Class Book

```
1 public class Publisher {
2     private String name;
3
4     public void setName(String name) {
5         this.name = name;
6     }
7     public String getName() {
8         // TODO: Implement code of Publisher.name()
9         return this.name;
10    }
11 }
```

Listing 2.13: Java Class Publisher

2.4.5.3 Using Xtend in Xpand Projects

Xtend is part of the Xpand project. It is a powerful language used basically for M2M transformation. However, Xtend can also help in formulating readable Xpand templates by defining reusable operations that access the source model and retrieve data from it. For example, the Xpand template in Listing 2.14 used *FOREACH* block to get the parameters of the operations.

```

1 «FOREACH ownedParameter.reject(e|e.direction.toString().toLowerCase().
2 endsWith("return")) AS p SEPARATOR ", "»
3
4   «p.type.name-» «p.name-»
5
6 «ENDFOREACH-»

```

Listing 2.14: Sample FOREACH in Xpand

The Xpand code in Listing 2.14 must be used wherever it is necessary to retrieve the parameters of operations. In order to improve the quality of the Xpand templates and sustain their reusability, it is possible to define a simple function with Xtend and use it inside Xpand templates by only invoking its name in the appropriate location in the templates. The Xtend operation can be formulated as shown in Listing 2.15:

```

1 List[uml::Parameter] getParameters(uml::Operation this):
2   ownedParameter.reject(e|e.direction.toString().toLowerCase().endsWith("return"));

```

Listing 2.15: Xtend Sample Operation

A reference to the function *getParameters* can replace the expression in the *FOREACH* block, i.e. (*ownedParameter.reject(e|e.direction.toString().toLowerCase().endsWith("return"))*) without changing the output. The Xpand template file must declare the extension of the Xtend file using the *EXTENSION* keyword followed by the name of the Xtend file as shown in Listing 2.16:

```

1 «EXTENSION myExtensionFiles::JavaExtensions»

```

Listing 2.16: Using Extensions in Xpand

Chapter 3

UML Profile for Web Services

The proposed **UML** profile in this thesis is defined after investigating different types of real Web service implementations. The investigations tried to figure out the main characteristics of Web services. Therefore, recognising the main elements of Web services, which form its novelty among the different types of software applications. This means, we need to identify all the existing elements and those, which are candidate to be existing in any possible implementation of a Web service. The UML Profile for Web Services (**UP4WS**) represents the core of the modelling of Web services, and the basis, on which the code generation process will rely, since the proposed code generation process and the transformation rules will handle directly the stereotypes of the **UP4WS**. The **UP4WS** defines two categories of **UML** extensions. The first category contains the basic extensions, which must be found in any Web service application. The second category includes the stereotypes that make Web services executable by enabling the generation of a complete and executable source code from the model elements applying those stereotypes. The target output from the **UP4WS** are:

- Source code in Java
- Configuration files for the Apache Axis2/Java Web service engine
- Documentation files

The **UP4WS** is defined in terms of stereotypes and, where necessary appropriate constraints and tagged values are added to complete the relevant semantics. In this thesis, two types of views on Web services have been considered. The service provider side, and service client side. The service client side view has been taken into consideration, since a Web service may invoke another Web service to provide a comprehensive response. In this case, the Web service plays the role of service

provider and service client at the same time. This leads to the necessity to implement both sides, and thus in the definition of extensions for both of them.

3.1 Web Services Basic Extensions

The basic extensions are mandatory extensions and must be specified for any Web service implementation, since they represent the necessary parts for Web services. Each extension is described by introducing the following characteristics:

1. UP4WS Stereotype
2. Metaclass
3. Semantics
4. Constraints
5. Transformation

In order to avoid the complexity of the **OC**L syntax, natural language will be used to formulate the constraints. As mentioned before, the proposed profile has been defined after thorough investigations of different types of Web services implementations. As a result of these investigations, the following observations have been recognised:

- Each Web service application comprises *service provider side*, and *service client side*.
- The service provider side must have a *service object* representing the different services provided by the Web service. These services can be represented in terms of *operations or methods*.
- The methods or operations in the service object manipulate data stored in other objects, i.e. *data objects*. These objects provide different data elements, i.e. *data variables or attributes*.
- The service client side contains always a *client object* that invokes the services provided by the Web service and represented in the *service object*. The execution of the Web service is achieved by executing the *client object*. Therefore, it must contain an execution mechanism (e.g. *main method* in Java or C). Our assumption is to define two objects at the service client side. The *proxy client object*, which is accessed by another *client object*. The *proxy client object* implements the services from the client's perspective, while the *client object* contains the execution mechanism of the Web service.

From the above mentioned observations, the following stereotypes have been defined to reflect the mandatory UML extensions for Web services:

- «WebService» to represent the service object.
- «ProxyMethod» to represent the services provided by the Web service in the service provider and the service client sides.
- «DataContainer» to represent the concrete data of the Web service.
- «DataElement» to represent the data elements as attributes.
- «ProxyImplementation» to represent the proxy client object that implements the services from the client's point of view.
- «Client» to represent the client object that contains the execution mechanism of the Web service.
- «ClientMain» to represent the method that enables executing the Web service in the client object (e.g. main method).

Figure 3.1 gives an overview of the basic extensions in the UP4WS, and how they will be applied by the model elements. The figure describes a weather Web service, which applies the UP4WS. The weather Web service model includes all mandatory extensions proposed in the UP4WS as follows:

- *WeatherService* object applying the «WebService» stereotype.
- *setWeather* and *getWeather* methods applying the «proxymethod» stereotype.
- *Weather* object applying the «DataContainer» stereotype.
- *temp* and *wind* attributes applying the «DataElement» stereotype.
- *WeatherServiceClient* object applying the «ProxyImplementation» stereotype.
- *WeatherServiceClientMain* applying the «Client» stereotype.
- *main* method applying the «ClientMain» stereotype.

In the following, the proposed basic UML extensions for Web services, which have been defined as mandatory extensions for any Web services application will be presented in more detail.

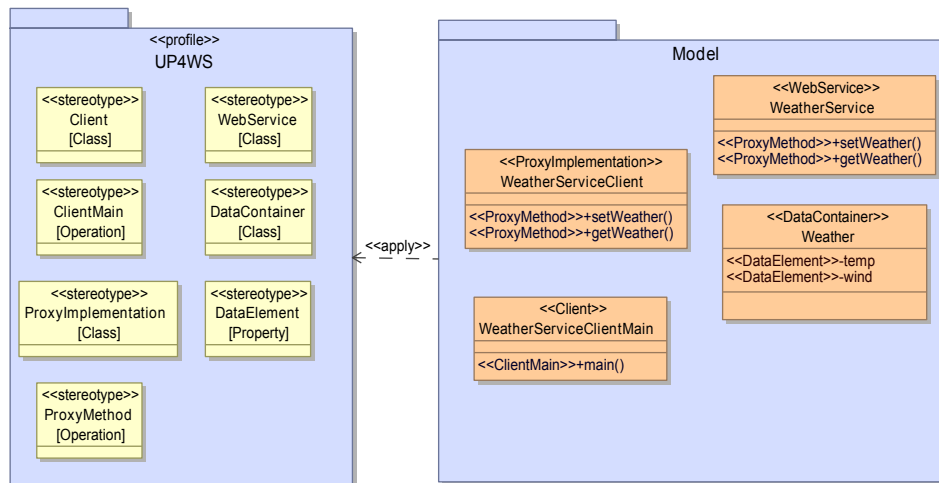


Figure 3.1: Basic Extensions and Their Usage

3.1.1 WebService

The «WebService» stereotype represents the Web service interface in the implementation language. The «WebService» stereotype must be considered in any Web service application, since it includes the implementation of services provided by the Web service at the service provider side in terms of operations. The *WeatherService* object, that appears in Figure 3.1 applies the «WebService» stereotype. The *WeatherService* represents a Web service that provides weather information. It includes the methods *setWeather* and *getWeather*, which in turn correspond to the services provided by the Web service.

- UP4WS Stereotype: «WebService»
- Metaclass: Class (from *Classes::Kernel 7.3.7 [UML10b]*)
- Semantics: The «WebService» stereotype is applied by the object that contains the operations representing the services, which the Web service provides. Each operation with a public visibility in the «WebService» object is accessible and can be invoked by the service requester or service client.
- Constraints: The following constraint(s) must be considered:
 - Only one object in the service provider side can apply the «WebService» stereotype.
- Transformation: For the «WebService» stereotype, a file representing the service object is to be generated. Inside this file, all relevant contents, such

as attributes and operations are to be generated by invoking the corresponding parts in the transformation rules, which are responsible for generating them. Import statements and package declarations must also be specified.

3.1.2 DataContainer

The «DataContainer» stereotype represents the data variables processed by the Web service. In the weather Web service example in Figure 3.1, one object applying the «DataContainer» stereotype can be seen (i.e. *Weather*). It is also possible to decompose the *Weather* object into two objects *Temperature* and *Wind*, where each object will have its own attributes. The decision is taken by the developers according to the requirements and their design preferences.

- UP4WS Stereotype: «DataContainer»
- Metaclass: Class (from *Classes::Kernel 7.3.7 [UML10b]*)
- Semantics: The object applying the «DataContainer» stereotype is located at the service provider side, and represents the entities and objects corresponding to the variable and processable data needed in order to enable the Web service to offer its functionality.
- Constraints: The following constraint(s) must be considered:
 - At least one object must apply «DataContainer» stereotype.
- Transformation: Analogous to the «WebService» stereotype, a file is to be generated for each object extending the «DataContainer» stereotype.

3.1.3 DataElement

The «DataElement» stereotype represents the data elements that can be found in any object in the Web service. It is basically defined to represent the data elements in the objects applying the «DataContainer» stereotype. However, it can be used in other objects, since it has similar semantics. Figure 3.1 shows two attributes applying the «DataElement» stereotype, i.e. *temp*, *wind* in *Weather* object.

- UP4WS Stereotype: «DataElement»
- Metaclass: Property (from *Classes::Kernel 7.3.44 [UML10b]*)
- Semantics: Any property applying the «DataElement» stereotype represents data definitions with "processable data" or properties, which can normally be accessed via setter and getter methods.
- Constraints: The following constraint(s) must be considered:

- Visibility is always *private*
- Transformation: For the «DataElement» stereotype, an attribute declaration is to be generated. Different scenarios should be taken into consideration. For example; visibility, type, abstract, default value, default value type.

3.1.4 ProxyImplementation

The «ProxyImplementation» stereotype is applied by the object containing the implementation of the services from the client's point of view. This means, the object, which applies the «ProxyImplementation» stereotyped is located at the service client side. In the case of the weather Web service in Figure 3.1, the object applying «ProxyImplementation» stereotype implements the services *setWeather* and *getWeather* from the client's point of view.

- UP4WS Stereotype: «ProxyImplementation»
- Metaclass: Class (*from Classes::Kernel 7.3.7 [UML10b]*)
- Semantics: The object applying the «ProxyImplementation» stereotype is located at the client side. It is a proxy client object that implements the services from the client's perspective. It implements only the public methods of the server side object applying the «WebService» stereotype.
- Constraints: The following constraint(s) must be considered:
 - Only one object can apply the «ProxyImplementation» stereotype.
 - The object applying the «ProxyImplementation» stereotype must have the same name of the object applying the «WebService» stereotype concatenated with the word *Client*. For example, if the object applying the «WebService» stereotype has the name *MyWebService*, the name of the object applying «ProxyImplementation» stereotype must have the name *MyWebServiceClient*. These constraints are important to execute the Web service according to the assumption of this thesis.
- Transformation: A file declaration for the object applying the «ProxyImplementation» stereotype is to be generated.

3.1.5 ProxyMethod

The «ProxyMethod» stereotype is representing the services provided by the Web service in the service provider and the service client sides. This means, they appear in the objects applying the stereotypes «WebService» in the service provider side, and «ProxyImplementation» in the service client side. However, they are

implemented differently in both sides. Figure 3.1 shows two methods applying the «ProxyMethod» stereotype, *setWeather* and *getWeather*, which appear in the *WeatherService* and *WeatherServiceClient* objects.

- UP4WS Stereotype: «ProxyMethod»
- Metaclass: Operation (from *Classes::Kernel 7.3.36 [UML10b]*)
- Semantics: The «ProxyMethod» stereotype is applied by the public attributes in the object applying the stereotype «WebService» and all properties in the object applying the «ProxyImplementation» stereotype.
- Constraints: The following constraint(s) must be considered:
 - Visibility must be public.
- Transformation: The «ProxyMethod» stereotype is applied by the methods or operations found in the objects applying the «WebService» or «ProxyImplementation» stereotypes. Therefore, the relevant method declaration is to be generated for each method applying the «ProxyMethod» stereotype.

3.1.6 Client

Logically, each Web service is invoked by one service client at least. For this purpose, the «Client» stereotype is defined to test the execution of the Web service. The object applying the «Client» stereotype includes the mechanism to execute the Web service. The weather Web service in Figure 3.1 includes one object (*WeatherServiceClientMain*) applying the «Client» stereotype. The Web service can be executed by running the *WeatherServiceClientMain* object, since it contains the execution mechanism (e.g. main method).

- UP4WS Stereotype: «Client»
- Metaclass: Class (from *Classes::Kernel 7.3.7 [UML10b]*)
- Semantics: The object, which applies the «Client» stereotype contains the execution mechanism for the Web service (i.e. main method). It invokes the methods found in the object applying the stereotype «ProxyImplementation» to execute the Web service.
- Constraints: The following constraint(s) must be considered:
 - The object applying the «Client» stereotype must have the same name as the object applying the «ProxyImplementation» stereotype concatenated with the word *Main*. For example, if the object, to which the «ProxyImplementation» stereotype is applied has the name *MyWebServiceClient*, the name of the object applying the «Client» stereotype

must have the name *MyWebServiceClientMain*. The weather Web service example in Figure 3.1 shows a realisation of the name constraint.

- Only one object can apply the «Client» stereotype (More than one client can invoke the Web service. This constraint is only added to make the illustration of the Web service code generation and its execution understandable to the readers).
- Transformation: For the «Client» stereotype, an object file is to be generated. This file will contain the execution mechanism.

3.1.7 ClientMain

The «ClientMain» stereotype is defined to reflect the mechanism required for the execution of the Web service. This means that it is applied by the method, which is responsible for executing the application. In the weather Web service example in Figure 3.1, the main method in the *WeatherServiceClientMain* object applies the «ClientMain» stereotype.

- UP4WS Stereotype: «ClientMain»
- Metaclass: Operation (*from Classes::Kernel 7.3.36 [UML10b]*)
- Semantics: The operation applies the «ClientMain» stereotype represents the main method in the client object, which is usually applying the «Client» stereotype.
- Constraints: The following constraint(s) must be considered:
 - The method applying the «ClientMain» stereotype is only found in the object applying the «Client» stereotype.
- Transformation: For the «ClientMain» stereotype, a main method declaration is to be generated.

3.2 Making Web Services Executable

In order to enable the generation of a complete source code and configuration files that make Web services executable, an additional set of UML extensions is required in the UP4WS. The new set of extensions enables the generator to access the stereotyped elements in the UML model and generate output according to their specifications. The extensions in this respect are classified into two categories, i.e. extensions that enable the generation of Web services behaviour from state machines, and the extensions that enable completing the source code (e.g. attribute declarations). These extensions are not Web services specific. Therefore, they are defined separately.

3.2.1 Executable State Machines

In order to execute a Web service, its behaviour must be specified. Therefore, additional extensions are required to reflect this behaviour. Since state machines have been chosen to represent the behaviour of Web services, additional extensions to be applied by state machine diagrams and their elements, such as states and transitions, have to be specified. The thesis distinguishes between two types of state machine extensions:

- «ObjectStateMachine» used for the implementation of the behaviour of any object that has multiple states, and
- «OperationStateMachine» used for the implementation of the class operations that change the behaviour of the object.

In order to complete the implementation of each stereotype, additional extensions applied by the elements of each state machine have to be specified (Sections 3.2.1.1 and 3.2.1.3).

Figure 3.2 shows a state machine diagram that represent the behaviour of a *Light* object. The state machine applies the «ObjectStateMachine» stereotype and includes two states to reflect the light states *LightOn* or *LightOff* triggered by two events in turn *turnOn()* and *trunOff()*. It includes the «ObjectState», «ObjectTransition», and «ObjectFinalState» stereotypes. State machines applying the «ObjectStateMachine» stereotype are used to specify the behaviour of objects that can have multiple states, which in turn enable full code generation.

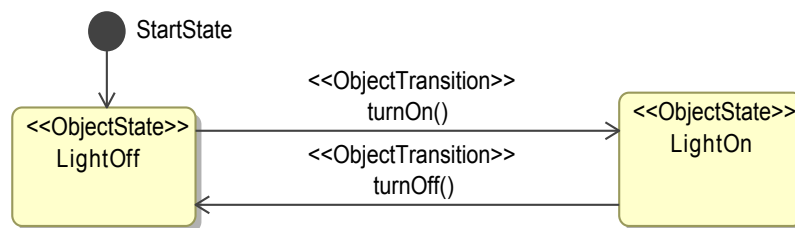


Figure 3.2: «ObjectStateMachine» Stereotype Implementation

3.2.1.1 ObjectStateMachine

The «ObjectStateMachine» stereotype is defined to enable generating the behaviour implementation of the data objects. If one of the data objects has a behaviour, the state machine representing this behaviour should apply the «ObjectStateMachine» stereotype. Other stereotypes that enable its implementation

shall also be included. Those stereotypes include «ObjectState», «ObjectTransition» and «ObjectFinalState» as shown in Figure 3.2.

- UP4WS Stereotype: «ObjectStateMachine»
- Metaclass: StateMachines (*from StateMachines::BehaviorStateMachines 15.3.12 [UML10b]*)
- Semantics: The «ObjectStateMachine» stereotype is to be applied by the state machine representing the behaviour of an object.
- Constraints: The following constraint(s) must be considered:
 - The name of the state machine applying the «ObjectStateMachine» stereotype must have the name of the object, to which it is assigned concatenated with the word *Status*. For example, if the object name is *Light*, the state machine name must be *LightStatus*.
 - The State machine applying «ObjectStateMachine» stereotype can describe the behaviour of any object applying the «DataContainer» stereotype.
- Transformation: The «ObjectStateMachine» stereotype represents the behaviour of any object applying the «DataContainer» stereotype. There are several ways to implement the state machine diagram and generate target source code. Samek [Sam02] described four possible methods for implementing state machines:
 1. The nested switch statement.
 2. The state table.
 3. The object-oriented state design pattern.
 4. A combination of the above.

In this thesis, the nested switch statement approach will be applied. It can be described in the following steps as described in [Sam02, VKEH06]:

1. create an enumeration object for all states.
2. create an enumeration object for all events.
3. define a variable to store the current state in the state machine.
4. specify a function for events, which trigger the transition. It shall do the following:
 - a) check all states to find the current state,
 - b) check all transitions corresponding to the event,
 - c) if the current state and the corresponding transition are found,s

- i. exit the current state by executing its exit action,
 - ii. the target state becomes the current state,
 - iii. enter the target state by executing its entry action,
 - iv. return.
5. An exception handler can be added.

The stereotypes «ObjectState», «ObjectTransition», and «ObjectFinalState» are parts of the implementation of the «ObjectStateMachine» stereotype.

3.2.1.1.1 ObjectState

As part of the implementation of the «ObjectStateMachine» stereotype, the «ObjectState» stereotype is defined to represent the different states of the object, to which the owning state machine is assigned. Figure 3.2 shows two states, i.e. *LightOff* and *LightOn* applying the «ObjectState» stereotype.

- UP4WS Stereotype: «ObjectState»
- Metaclass: State (*from StateMachines::BehaviorStateMachines, StateMachines::ProtocolStateMachines 15.3.11 [UML10b]*)
- Semantics: The «ObjectState» stereotype is applied by the states in the state machine that applies the «ObjectStateMachine» stereotype. Each state represents a possible status of the object.
- Constraints: The following constraint(s) must be considered:
 - * No constraints specified.
- Transformation: see Section 3.2.1.1

3.2.1.1.2 ObjectTransition

The «ObjectTransition» stereotype is defined to complete the implementation of the «ObjectStateMachine» stereotype. The «ObjectTransition» stereotype. It represents the transitions between the states. Figure 3.2 shows how the «ObjectTransition» stereotype participates in the implementation of any state machine applying the «ObjectStateMachine» stereotype.

- UP4WS Stereotype: «ObjectTransition»
- Metaclass: Transition (*from StateMachines::BehaviorStateMachines 15.3.14 [UML10b]*)
- Semantics: The «ObjectTransition» stereotype is applied by the transitions in the state machine that applies the stereotype «ObjectStateMachine». It represents the transitions between the states in this state machine.

- Constraints: The following constraint(s) must be considered:
 - * No constraints specified.
- Transformation: see Section 3.2.1.1

3.2.1.1.3 ObjectFinalState

The «ObjectFinalState» stereotype is part of the implementation of the «ObjectStateMachine» stereotype, and enables the completion of its implementation, as shown in Figure 3.2.

- UP4WS Stereotype: «ObjectFinalState»
- Metaclass: FinalState (from *StateMachines::BehaviorStateMachines 15.3.2 [UML10b]*)
- Semantics: The «ObjectFinalState» stereotype is applied by the final state in the state machine that applies the stereotype «ObjectStateMachine», and represents the final state of this state machine.
- Constraints: The following constraint(s) must be considered:
 - * No constraints specified.
- Transformation: see Section 3.2.1.1

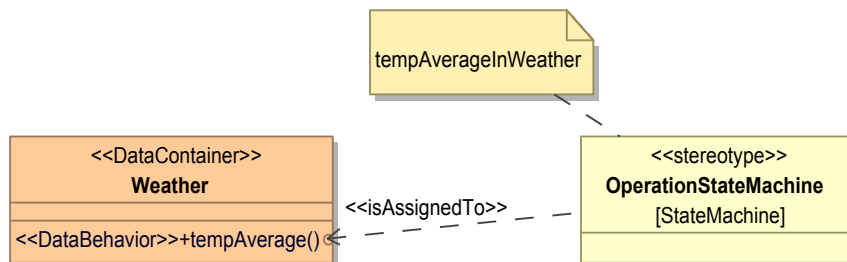


Figure 3.3: «DataBehavior» Stereotype

3.2.1.2 DataBehavior

The «DataBehavior» stereotype represents the methods processing the data variables, excluding setter and getter methods. These methods add behaviour to the owning object, and thus change some values of that object. A behavioural diagram (e.g. a state machine) could be assigned to any method applying the «DataBehavior» stereotype in order to enable its implementation. The assigned state machine must apply the «OperationStateMachine» stereotype. The code generated from the state machine will correspond to the implementation of the method. Figure 3.3 illustrates an example for the «DataBehavior» stereotype.

The figure shows a state machine assigned to the *tempAverage* operation. The state machine *tempAverageInWeather* that applies the «OperationStateMachine» stereotype, while the operation *tempAverage* applies the «DataBehavior» stereotype. Figure 3.4 extends the example in Figure 3.3 by showing the usage of the state machine to generate the implementation of the objects methods. It presents a state machine diagram that applies the «OperationStateMachine» stereotype. It shows how the elements of the states *entry*, *do*, and *exit* can be used to generate the implementation of the operations. The «OperationState», and «OperationTransition» are parts of the implementation of the owning state machine.

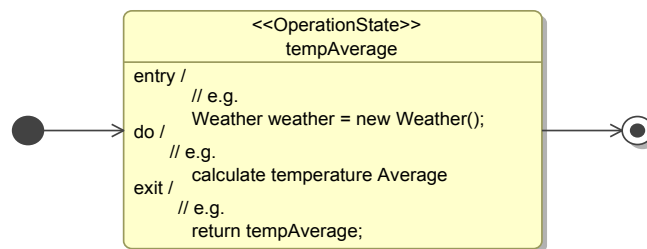


Figure 3.4: «OperationStateMachine» Stereotype Implementation

- UP4WS Stereotype: «DataBehavior»
- Metaclass: Operation (from *Classes::Kernel 7.3.36 [UML10b]*)
- Semantics: The operation applying the «DataBehavior» stereotype represents any operation that adds behaviour to its owning object.
- Constraints: The following constraint(s) must be considered:
 - No constraints specified.
- Transformation: For the «DataBehavior» stereotype, an operation signature is to be generated. The possible scenarios must be taken into consideration, such as visibility, return type, name, parameters list, and type of each parameter.

3.2.1.3 OperationStateMachine

The «OperationStateMachine» stereotype is defined to enable the generation of the implementation of the operations. As parts of the «OperationStateMachine» stereotype, the «OperationState» and «OperationTransition» stereotypes have been defined. Figure 3.4 shows an example for the «OperationStateMachine» stereotype, where it is assigned to an operation called *tempAverage*.

- UP4WS Stereotype: «OperationStateMachine»

- Metaclass: *StateMachines* (from *StateMachines::BehaviorStateMachines* 15.3.12 [UML10b])
- Semantics: The «OperationStateMachine» stereotype can be applied by any state machine describing the behaviour of an object, and is assigned to one of its owned operations. The behaviour remains owned by the object, although it is assigned to the operation.
- Constraints: The following constraint(s) must be considered:
 - The state machine applying the «OperationStateMachine» stereotype must have the same name of the operation, to which it is assigned, concatenated with the word *In* and the name of the object owning this operation. In the example in Figure 3.3, if the operation name is *tempAverage* and the owning object name is *Weather*, the state machine assigned to this operation must have the name *tempAverageInWeather*.
- Transformation: For the «OperationStateMachine» stereotype, the implementation of any operation, to which this state machine is assigned, is to be generated. The implementation of the «OperationStateMachine» stereotype includes the implementation of the stereotypes «OperationState» and «OperationTransition» stereotypes.

3.2.1.3.1 OperationState

The «OperationState» stereotype is defined to enable the completion of the implementation of the state machine applying the «OperationStateMachine» stereotype.

- UP4WS Stereotype: «OperationState»
- Metaclass: *State* (from *StateMachines::BehaviorStateMachines*, *StateMachines::ProtocolStateMachines* 15.3.11 [UML10b])
- Semantics: The «OperationState» stereotype is applied to the states in the state machine that applies the «OperationStateMachine» stereotype.
- Constraints: The following constraint(s) must be considered:
 - No constraints specified.
- Transformation: The state applying the «OperationState» stereotype form the implementation of the operation, to which the owning state machine is assigned. The items *entry*, *doActivity*, and *exit* shall correspond to the implementation, where the *entry* item represents initialisations, the *doActivity* item represents the pieces of code that follow initialisations, and the *exit* item represents return statements if any.

3.2.1.3.2 OperationTransition

The «OperationTransition» stereotype is defined to complete the implementation of the «OperationStateMachine» stereotype.

- UP4WS Stereotype: «OperationTransition»
- Metaclass: Transition (*from StateMachines::BehaviorStateMachines 15.3.14 [UML10b]*)
- Semantics: The «OperationTransition» stereotype is applied by the transitions in the state machine that applies the stereotype «OperationStateMachine».
- Constraints: The following constraint(s) must be considered:
 - No constraints specified.
- Transformation: see Section 3.2.1.3

3.2.2 Auxiliary Extensions

Another set of extensions is defined to enable the generation of specific implementation constructs. For example, attributes have different types of declarations and in order to define the transformation rules that enable the generation of those different attribute declarations, different types of extensions have to be specified, for example, the «Enumerizable» stereotype is defined to represent the attributes that have the type of an enumeration class, while the «Initializable» stereotype is defined to enable the generation of any attribute that should be initialised.

3.2.2.1 Initializable

The «Initializable» stereotype is defined in order to enable the implementation of attributes that have or should have initial values. The implementation of the «Initializable» stereotype is actually an extension of the implementation of the «DataElement» stereotype.

- UP4WS Stereotype: «Initializable»
- Metaclass: Property (*from Classes::Kernel 7.3.44 [UML10b]*)
- Semantics: Initializable is any property with an initial value.
- Constraints: The following constraint(s) must be considered:
 - The property applying «Initializable» stereotype must also apply the stereotype «DataElement».
 - An initial value must be specified for the property.

- Transformation: For the «Initializable» stereotype, an attribute declaration indicating the initial value of the attribute is to be generated. The «Initializable» stereotype is always applied to the attribute applying also the stereotype «DataElement» and can not be applied independently.

Figure 3.5 shows the usage of the «Initializable» stereotype. It shows that any attribute applying the «Initializable» stereotype must also apply the «DataElement» stereotype. In this case, the *weather* attribute has to be initialised.

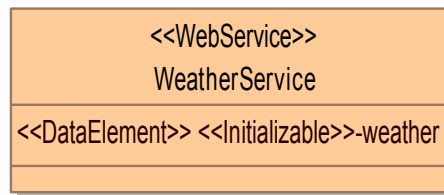


Figure 3.5: «Initializable» Stereotype Example

3.2.2.2 Enumerizable

The «Enumerizable» stereotype is defined to represent any attribute that has a type of an enumeration object in the model. Since the implementation of such attributes is unique, the «Enumerizable» stereotype has been defined separately.

- UP4WS Stereotype: «Enumerizable»
- Metaclass: Property (*from Classes::Kernel 7.3.44 [UML10b]*)
- Semantics: Enumerizable is any property that has a type of an enumeration object.
- Constraints: The following constraint(s) must be considered:
 - No constraints specified.
- Transformation: The «Enumerizable» stereotype is dedicated to the attributes with a type corresponding to an enumeration object. Therefore, the relevant attribute declaration considering the constraint is to be generated.

Figure 3.6 shows the usage of the «Enumerizable» stereotype. It shows that the *wind* attribute is of type *WindDirection*, which is an enumeration class and can have one of four enumeration literal values, i.e. *east*, *west*, *south*, and *north*.

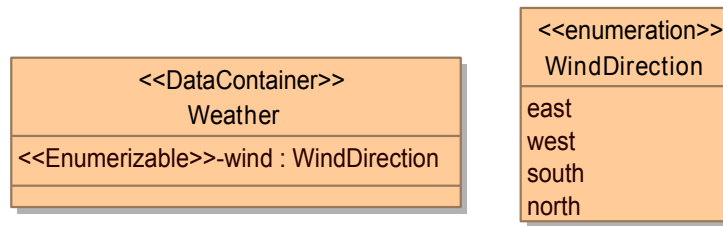


Figure 3.6: «Enumerizable» Stereotype Example

3.3 Profile Implementation

The profile implementation in this thesis is meant to represent the model transformation and the code generation process, which includes defining the transformation rules for transforming the model elements into the Java source code, configuration files, and documentation files. Each element in the source model applying any stereotype defined in the **UP4WS** is eligible to be accessed by the generator and thus influences the output files for deploying and implementing the Web service corresponding to the source model.

3.3.1 Model Transformation and Code Generation

The model transformation and code generation process aims at producing the target output from the **UML** model. Figure 3.7 gives an overview of the entire model transformation process. It shows the **UP4WS**, which is applied by a **UML** model. The model is then exported to the generator, which applies the transformation rules on it in order to generate the relevant output.

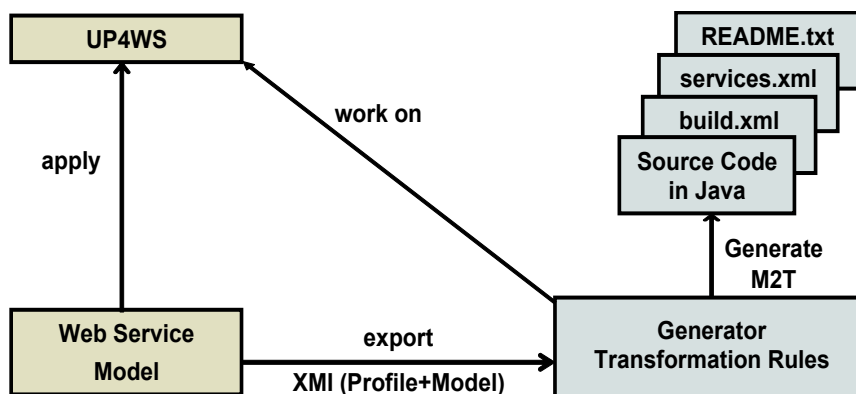


Figure 3.7: Model Transformation and Code Generation Process

The generator produces three different types of output:

Stereotype Name	Template File Name
«WebService»	CommonTemplate and WebService
«DataContainer»	CommonTemplate and DataContainer
«ProxyImplementation»	CommonTemplate and ProxyImplementation
«Client»	CommonTemplate and Client

Table 3.1: Mapping Between Template Files and Stereotypes

- Source code in Java (i.e. service provider and client sides),
- Configuration files (i.e. *build.xml* and *services.xml* files),
- *README.txt* file, which provide instructions on how to run the Web service.

The Xpand transformation language will be used for transforming the UML model into the target output files (i.e. source code and the configuration files for the platform). The Xpand transformation rules are defined in terms of templates distributed in several template files. Each template is responsible for generating the code corresponding to specific elements that applies one stereotype from the UP4WS in the UML model from the UP4WS in the UML model. It is also possible that more than one stereotype can be applied by the same element in the model. Each template file starts normally with *IMPORT* and *EXTENSION* statements and contains one or more templates. In the following, a detailed description for each template file and its contents is presented. The templates and the owning template files are used to generate the Java source code and the associated configuration and documentation files. The Xtend language will be used to define some operations, which can be used inside Xpand templates. Table 3.1 shows a mapping between the template files and the stereotypes. The right column shows the template files which contain the templates that generate the implementation for the stereotypes in the left column. The left column correspond to the stereotypes, from which all possible objects files can be generated.

3.3.1.1 CommonTemplates Template File

The CommonTemplates template file contains all the templates that can be invoked by the templates inside the same or other template files. These templates can be used in any other implementation of any application in Java.

3.3.1.1.1 createImport Template

The create Import template (Listing 3.1) is responsible for generating the import statements in the files of the objects.

```

1 «DEFINE createImport FOR uml::Type»
2   import «packageName()–».«"*"–»;
```

```
3 «ENDDEFINE»
```

Listing 3.1: createImport Template

3.3.1.1.2 createDataElement Template for «DataElement»

The createDataElement template (Listing 3.2) generates the declaration of any attribute applying the «DataElement» stereotype. Since the «Initializable» stereotype can be applied only to attributes applying the «DataElement» stereotype, the «Initializable» stereotype is implicitly included in the createDataElement template. The template implements different possible scenarios for the declaration of any attribute, such as type, default value, and type of the default value.

```
1 «DEFINE createDataElement FOR UP4WS::DataElement»
2   «IF defaultValue == null && metaType.name
3     != "UP4WS::DataElement,UP4WS::Initializable"»
4     «visibility» «type.name» «name»;
5   «ELSE»
6     «IF defaultValue == null && metaType.name
7       == "UP4WS::DataElement,UP4WS::Initializable"»
8       «visibility» «type.name» «name» = new «type.name»();
9     «ENDIF»
10    «IF defaultValue != null && type.name == "String"»
11      «visibility» «type.name» «name» = "«value()»";
12    «ELSE»
13      «IF defaultValue != null && type.name != "String"»
14        «visibility» «type.name» «name» = «value()»;
15      «ENDIF»
16    «ENDIF»
17  «ENDIF»
18 «ENDDEFINE»
```

Listing 3.2: createDataElement Template

3.3.1.1.3 Enumerizable Template for «Enumerizable»

The Enumerizable template (Listing 3.3) is responsible for generating the declaration of the attributes that apply the «Enumerizable» stereotype. The «Enumerizable» stereotype is dedicated to the attributes that have an enumeration object as a type.

```
1 «DEFINE Enumerizable FOR UP4WS::Enumerizable»
2
3   «IF defaultValue == null-»
4     «visibility» «type.name» «name-»;
5   «ELSE»
6     «visibility» «type.name» «name» =
7     «type.name».«value().toUpperCase()-»;
8   «ENDIF-»
9
10 «ENDDEFINE»
```

Listing 3.3: Enumerizable Template

3.3.1.1.4 gettersAndSetters Template for «DataElement» and «Enumerizable»

The `gettersAndSetters` template (Listing 3.4) creates the getter and setter methods for the attributes, to which one of these stereotypes is applied. It also checks whether the visibility of the attribute is *private*.

```

1 «DEFINE gettersAndSetters FOR UP4WS::DataElement»
2 «IF visibility.toString() == "private"»
3   public void set«name.toFirstUpper()»
4     («type.name» «name.toFirstLower()»){
5     this.«name» = «name.toFirstLower()»;
6   }
7   public «type.name» get«name.toFirstUpper()» (){
8     return this.«name»;
9   }
10 «ENDIF»
11 «ENDDEFINE»

```

Listing 3.4: `gettersAndSetters` Template

3.3.1.1.5 methodTpl Template for «DataBehavior»

The `methodTpl` template (Listing 3.5) generates the declaration and signature of any operation applying the «DataBehavior» stereotype. The implementation of this method is generated by the templates generating the source code for the state machines that applies the «OperationStateMachine» stereotype.

```

1 «DEFINE methodTpl FOR UP4WS::DataBehavior»
2 «visibility» «IF isStatic» static «ENDIF» «IF type==null» void
3 «ELSE» «type.name» «ENDIF»
4 «name» («FOREACH getParameters()
5 AS p SEPARATOR ", "»«p.type.name.toJava()» «p.name»«ENDFOREACH»){
6 «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
7 (UP4WS::OperationStateMachine) AS wssm-»
8 «IF name+"In"+owner.name == wssm.name-»
9 «EXPAND implementState(wssm) FOREACH
10 wssm.eAllContents.typeSelect(UP4WS::OperationState)-»
11 «ENDIF-»
12 «ENDFOREACH-»
13 }
14 «ENDDEFINE»

```

Listing 3.5: `methodTpl` Template

3.3.1.1.6 Events Template for «ObjectStateMachine»

The Events template (Listing 3.6) generates an enumeration class containing all events that appear in the state machine that applies the «ObjectStateMachine» stereotype. The template also checks whether the state machine is representing the behaviour of an object that applies the «DataContainer» stereotype.

```

1 «DEFINE Events FOR UP4WS::ObjectStateMachine»
2 «IF owner.metaType.name == "UP4WS::DataContainer"»
3 «FILE eventsEnumFileName() myDataOutlet-»
4 package «packageName(this)»;
5 public enum «eventsEnumName()» {
6     «FOREACH events().constantName().toSet() AS s SEPARATOR ","-»
7     «s-»
8     «ENDFOREACH»
9 }
10 «ENDFILE»
11 «ENDIF»
12 «ENDDEFINE»

```

Listing 3.6: Events Template

3.3.1.1.7 States Template for «ObjectStateMachine»

The States template (Listing 3.7) generates an enumeration class containing all states that appear in any state machine applying the stereotype «ObjectStateMachine» and is assigned to any object, which applies the «DataContainer» stereotype.

```

1 «DEFINE States FOR UP4WS::ObjectStateMachine»
2 «IF owner.metaType.name == "UP4WS::DataContainer"»
3 «FILE statesEnumFileName()myDataOutlet-»
4 package «packageName(this)»;
5
6 public enum «statesEnumName()»{
7     «FOREACH states() AS s SEPARATOR ","-»
8     «s.constantName()»
9     «ENDFOREACH»
10 }
11 «ENDFILE»
12 «ENDIF»
13 «ENDDEFINE»

```

Listing 3.7: States Template

3.3.1.1.8 implementState Template for «OperationState»

The implementState template (Listing 3.8) generates the implementation of any method, to which the state machine owning this state is assigned.

```

1 «DEFINE implementState (UP4WS::OperationStateMachine sm)
2 FOR UP4WS::OperationState»
3 «IF entry != null-»
4     «entry.name-»
5 «ENDIF-»
6 «IF doActivity != null-»

```

```

7  «doActivity.name-»
8  «ENDIF»
9  «IF exit != null-»
10 «exit.name-»
11 «ENDIF-»
12 «ENDDEFINE»

```

Listing 3.8: implementState Template

3.3.1.2 WebService Template File

The WebService template file contains all the templates required for the generation of the object applying the «WebService» stereotype. The WebService template file contains a set of templates in order to enable the generation of the corresponding Java file.

3.3.1.2.1 serviceRoot Template for «WebService»

The serviceRoot template (Listing 3.9) invokes the template, which creates the Java file for the object applying the «WebService» stereotype.

```

1 «DEFINE serviceRoot FOR UP4WS::WebService»
2 «EXPAND createWebService»
3 «ENDDEFINE»

```

Listing 3.9: serviceRoot Template

3.3.1.2.2 createWebService Template for «WebService»

The createWebService template (Listing 3.10) is the core template in the WebService template file, since it generates the Java file for the object applying the «WebService» stereotype. The createWebService template invokes the following templates to generate the implementation of the Java file:

- createImport from CommonTemplates template file (Section 3.3.1.1.1).
- createDataElement from CommonTemplates template file (Section 3.3.1.1.2).
- Enumerizable from CommonTemplates template file (Section 3.3.1.1.3).
- methodTmpl from CommonTemplates template file (Section 3.3.1.1.5).
- gettersAndSetters from CommonTemplates template file (for «Enumerizable» and «DataElement») (Section 3.3.1.1.4).
- proxyMethodServiceTmpl from the WebService template file (Section 3.3.1.2.3).


```

1 «DEFINE createWebService FOR UP4WS::WebService»
2
3 «FILE name + ".java" myServiceOutlet»
4 package «packageName(this»);
5
6 «EXPAND CommonTemplates::createImport FOREACH usedTypes()»
7 «visibility-» class «name-»{
8 «EXPAND CommonTemplates::createDataElement FOREACH
9 eAllContents.typeSelect(UP4WS::DataElement)-»
10
11 «EXPAND CommonTemplates::Enumerizable FOREACH
12 eAllContents.typeSelect(UP4WS::Enumerizable)-»
13
14 «EXPAND CommonTemplates::methodTpl FOREACH
15 eAllContents.typeSelect(UP4WS::DataBehavior)-»
16
17 «EXPAND CommonTemplates::gettersAndSetters FOREACH
18 eAllContents.typeSelect(UP4WS::Enumerizable)-»
19
20 «EXPAND CommonTemplates::gettersAndSetters FOREACH
21 eAllContents.typeSelect(UP4WS::DataElement)-»
22
23 «EXPAND proxyMethodServiceTpl FOREACH
24 eAllContents.typeSelect(UP4WS::ProxyMethod)-»
25 }
26 «ENDFILE»
27 «ENDDEFINE»

```

Listing 3.10: createWebService Template

3.3.1.2.3 proxyMethodServiceTpl Template for «ProxyMethod»

The proxyMethodServiceTpl template (Listing 3.11) is responsible for generating the methods inside the Java file applying the «WebService» stereotype. It generates the signature for each method applying the «ProxyMethod» stereotype. The implementation of the methods will be generated by invoking the template *implementState* from the CommonTemplates template file.

```

1 «DEFINE proxyMethodServiceTpl FOR UP4WS::ProxyMethod»
2
3 «visibility-» «IF isStatic-» static «ENDIF-»
4 «IF type==null-» void «ELSE-» «type.name-» «ENDIF-» «name-»
5 («FOREACH getParameters().sortBy(e|e.name) AS p SEPARATOR ","-»
6 «p.type.name.toJava()-» «p.name-» «ENDFOREACH-»){
7 «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
8 (UP4WS::OperationStateMachine) AS wssm-»
9 «IF name+"In"+owner.name == wssm.name-»
10
11 «EXPAND CommonTemplates::implementState(wssm)
12 FOREACH wssm.checkStates()-»
13
14 «ENDIF-»
15 «ENDFOREACH-»
16 }
17 «ENDDEFINE»

```

Listing 3.11: proxyMethodServiceTpl Template

3.3.1.3 DataContainer Template File

The DataContainer template file contains all the templates needed for generating the Java files for the objects that apply the «DataContainer» stereotype. It contains a set of templates to generate the corresponding Java files and their implementations.

3.3.1.3.1 dataContainerRoot Template for «DataContainer»

The dataContainerRoot template (Listing 3.12) is developed to make the template createDataContainer reusable by invoking it where appropriate.

```

1 «DEFINE dataContainerRoot FOR UP4WS::DataContainer»
2   «EXPAND createDataContainer»
3 «ENDDEFINE»

```

Listing 3.12: dataContainerRoot Template

3.3.1.3.2 createDataContainer Template for «DataContainer»

The createDataContainer template (Listing 3.13) is the main template in DataContainer template file, since it generates a Java file declaration for each object applying the «DataContainer» stereotype. In the createDataContainer template, the following templates are invoked:

- createImport from CommonTemplates template file (Section 3.3.1.1.1).
- createDataElement for «DataElement» from CommonTemplates template file (Section 3.3.1.1.2).
- Enumerizable for «Enumerizable» from CommonTemplates template file (Section 3.3.1.1.3).
- methodTmpl for «DataBehavior» from CommonTemplates template file (Section 3.3.1.1.5).
- gettersAndSetters for «Enumerizable» and «DataElement» from CommonTemplates template file (Section 3.3.1.1.4).
- ObjectBehavior for «ObjectStateMachine» from the same template file (Section 3.3.1.3.3).
- executeTransitionForHandleEvent for «ObjectTransition» (Section 3.3.1.3.4)
- illegalTransitionHandler for «ObjectStateMachine» (Section 3.3.1.3.5)

```

1 «DEFINE createDataContainer FOR UP4WS::DataContainer»
2
3 «FILE name + ".java" myDataOutlet-»
4
5 package «packageName(this);
6
7 «EXPAND CommonTemplates::createImport FOREACH usedTypes()»
8
9 «visibility-» «IF isAbstract-» abstract «ENDIF-» class «name»
10 «IF !superClass.isEmpty-» extends «superClass.first().name-»«ENDIF-»
11 «IF this.interfaceRealisation.contract.size > 0-» implements
12 «FOREACH this.interfaceRealisation.contract AS i SEPARATOR ","-»
13 «i.name-»«ENDFOREACH-»«ENDIF-»{
14
15     «EXPAND CommonTemplates::createDataElement
16     FOREACH eAllContents.typeSelect(UP4WS::DataElement)-»
17
18     «EXPAND CommonTemplates::Enumerizable
19     FOREACH eAllContents.typeSelect(UP4WS::Enumerizable)-»
20
21     «EXPAND CommonTemplates::gettersAndSetters
22     FOREACH eAllContents.typeSelect(UP4WS::Enumerizable)-»
23
24     «EXPAND CommonTemplates::gettersAndSetters
25     FOREACH eAllContents.typeSelect(UP4WS::DataElement)-»
26
27     «EXPAND CommonTemplates::methodTpl
28     FOREACH eAllContents.typeSelect(UP4WS::DataBehavior)-»
29
30     «EXPAND classBehavior
31     FOREACH eAllContents.typeSelect(UP4WS::ObjectStateMachine)-»
32
33 }
34 «ENDFILE»
35 «ENDDEFINE»

```

Listing 3.13: createDataContainer Template

3.3.1.3.3 ObjectBehavior Template for «ObjectStateMachine»

The ObjectBehavior template (Listing 3.14) is responsible for generating the behaviour of any object applying the «DataContainer» stereotype. The behaviour is represented by state machine diagrams, and implemented using the nested switch based approach (Section 3.2.1.1). The ObjectBehavior template starts by checking that the state machine is assigned to an object applying the «DataContainer» stereotype. The implementation is done by generating a method named *handleEvent*. The nested switch statement will form the implementation of this method. The ObjectBehavior template invokes the following templates:

- executeTransitionForHandleEvent for «ObjectTransition» (Section 3.3.1.3.4).
- illegalTransitionHandler for «ObjectStateMachine» (Section 3.3.1.3.5)

```

1 «DEFINE classBehavior FOR UP4WS::ObjectStateMachine»
2
3 «IF name == getOwnerName(this)+"Status" &&
4 owner.metaType.name == "UP4WS::DataContainer"»
5
6 public void handleEvent(«eventsEnumName(this)» event){
7     switch (currentState) {
8         «FOREACH this.states().reject
9         (s|UP4WS::ObjectFinalState.isInstance(s)) AS s-»
10        case «s.constantName()»:
11            «FOREACH s.outTransitionsWithEventTrigger() AS t-»
12            «IF t.trigger.event !=null-»
13            «FOREACH t.trigger.event AS e-»
14            if (event == «e.eventId(this)») {
15                «EXPAND executeTransitionForHandleEvent(this) FOR t»
16            }
17            break;
18            }«ENDFOREACH»«ENDIF»«ENDFOREACH»
19            «EXPAND illegalTransitionHandler»
20        }
21    }
22 «ENDIF»
23 «ENDDEFINE»

```

Listing 3.14: ObjectBehavior Template

3.3.1.3.4 executeTransitionForHandleEvent Template for «Object Transition»

The executeTransitionForHandleEvent template (Listing 3.15) generates the pieces of the source code needed for executing the transition. It also checks the events triggering the transitions to move between the states. In the final state, it terminates.

```

1 «DEFINE executeTransitionForHandleEvent(UP4WS::ObjectStateMachine sm)
2 FOR UP4WS::ObjectTransition»
3
4 «IF UP4WS::ObjectState.isInstance(source) &&
5 ((UP4WS::ObjectState)source).exit!=null-»
6 «((UP4WS::ObjectState)source).exit.methodName()»();
7 «ENDIF-»
8 «FOREACH trigger.event AS e-»
9 «IF effect!=null-»
10 «effect.methodName()»();
11 «ENDIF»
12 «ENDFOREACH»
13 currentState = «target.stateId(sm)»;
14 «IF UP4WS::ObjectFinalState.isInstance(target)-»
15     terminated = true;
16 «ENDIF-»
17 «IF UP4WS::ObjectState.isInstance(target) &&
18 ((UP4WS::ObjectState)target).entry!=null-»
19 «((UP4WS::ObjectState)target).entry.methodName()»();
20 «ENDIF-»
21 «ENDDEFINE»

```

Listing 3.15: executeTransitionForHandleEvent Template

3.3.1.3.5 illegalTransitionHandler Template for «ObjectStateMachine»

The illegalTransitionHandler template (Listing 3.16) generates the exception handler inside the nested switch statement.

```

1 «DEFINE illegalTransitionHandler FOR UP4WS::ObjectStateMachine»
2   throw new IllegalStateException("Event "+event+"
3   for state "+currentState+" can not be handled");
4 «ENDDEFINE»

```

Listing 3.16: illegalTransitionHandler Template

3.3.1.4 Proxy Template File

The Proxy template file contains all the templates required for the generation of the Java file based on the stereotype «ProxyImplementation». The following templates are found inside the Proxy template file:

3.3.1.4.1 proxyRoot Template for «ProxyImplementation»

The proxyRoot template (Listing 3.17) invokes the template createProxyImpl (Section 3.3.1.4.2) for the same stereotype for reusability.

```

1 «DEFINE proxyRoot FOR UP4WS::ProxyImplementation»
2
3   «EXPAND createProxyImpl–»
4
5 «ENDDEFINE»

```

Listing 3.17: proxyRoot Template

3.3.1.4.2 createProxyImpl Template for «ProxyImplementation»

The createProxyImpl template (Listing 3.18) generates the Java file for the object applying the «ProxyImplementation» stereotype. For this purpose, it invokes the following templates:

- createImport template from CommonTemplates template file to retrieve all import statements from the model (Section 3.3.1.1.1)
- proxyMethodClientTmpl template for «ProxyMethod» from the same template file (Section 3.3.1.4.3).

```

1 «DEFINE createProxyImpl FOR UP4WS::ProxyImplementation»
2   «FILE name.toFirstUpper()+"_java" myClientOutlet–»
3   package «packageName(this)–»;
4
5   «EXPAND CommonTemplates::createImport FOREACH usedTypes()–»
6   import javax.xml.namespace.QName;
7   import org.apache.axis2.AxisFault;
8   import org.apache.axis2.addressing.EndpointReference;
9   import org.apache.axis2.client.Options;
10  import org.apache.axis2.rpc.client.RPCServiceClient;
11  «visibility» class «name.toFirstUpper()–» {

```

```

12     «EXPAND ProxyMethodClientTmpl
13     FOREACH eAllContents.typeSelect(UP4WS::ProxyMethod)-»
14     }
15     «ENDFILE»
16 «ENDDEFINE»

```

Listing 3.18: createProxyImpl Template

3.3.1.4.3 proxyMethodClientTmpl Template for «ProxyMethod»

The proxyMethodClientTmpl template (Listing 3.19) is responsible for generating the signatures and implementations of all methods that apply the «ProxyMethod» stereotype and having a *public* visibility.

```

1 «DEFINE ProxyMethodClientTmpl FOR UP4WS::ProxyMethod»
2
3 «IF getVisibility(this) == "public"-»
4     «visibility-» «IF isStatic-» static «ENDIF-»
5     «IF type==null-» void«ELSE-» «type.name-»«ENDIF-» «name-»
6     («FOREACH getParameters().sortBy(e|e.name) AS p SEPARATOR ", "-»
7     «p.type.name.toJava()» «p.name-»«ENDFOREACH-»,
8
9     RPCServiceClient rpcClient) throws AxisFault {
10    QName opGet = new QName("http://service.webservice.sample", "«name-»");
11    rpcClient.getOptions().setAction("urn:«name-»");
12    Object[] args = new Object[«count()-1»];
13    «FOREACH getParameters().sortBy(e|e.name) AS p ITERATOR i-»
14    args[«i.counter0»]= «p.name-»;
15    «ENDFOREACH»
16    «IF type.name == "void"-»
17    rpcClient.invokeRobust(opGet, args);
18    «ELSE-»
19    Class[] returnTypes = new Class[] { «type.name-».class };
20    Object[] response = rpcClient.invokeBlocking
21    (opGet, args, returnTypes);
22    «ENDIF-»
23
24    «FOREACH ((uml::Model)
25    this.eRootContainer).eAllContents.typeSelect
26    (UP4WS::OperationStateMachine) AS wssm-»
27    «IF name+"In"+owner.name == wssm.name-»
28    «EXPAND CommonTemplates::implementState(wssm)
29    FOREACH wssm.checkStates()-»
30    «ENDIF-»
31    «ENDFOREACH-»
32
33    «IF type.name == "void"-»
34    «ELSE-»
35    return («type.name») response [0];
36    «ENDIF-»
37    }
38 «ENDIF-»
39 «ENDDEFINE»

```

Listing 3.19: proxyMethodClientTmpl Template

3.3.1.5 Client Template File

The Client template file contains the templates that are responsible for the generation of the client Java file, which contains the main method. The file contains the following templates:

3.3.1.5.1 clientRoot Template for «Client»

The clientRoot template (Listing 3.20) is developed for reusability.

```

1 «DEFINE clientRoot FOR UP4WS::Client»
2
3   «EXPAND createClientImpl–»
4
5 «ENDDEFINE»

```

Listing 3.20: clientRoot

3.3.1.5.2 createClientImpl Template for «Client»

The createClientImpl template (Listing 3.21) generates the Java file for the object applying the «Client» stereotype. The createClientImpl template invokes the following templates:

- createImport template from CommonTemplates template file to retrieve all import statements from the model (Section 3.3.1.1.1)
- createException template to generate the behaviour, which is assigned to the main method (Section 3.3.1.5.3).

```

1 «DEFINE createClientImpl FOR UP4WS::Client»
2
3   «FILE name.toFirstUpper()+".java" myClientOutlet–»
4
5   package «packageName(this)–»;
6
7   «EXPAND CommonTemplates::createImport FOREACH usedTypes()–»
8
9   import javax.xml.namespace.QName;
10  import org.apache.axis2.AxisFault;
11  import org.apache.axis2.addressing.EndpointReference;
12  import org.apache.axis2.client.Options;
13  import org.apache.axis2.rpc.client.RPCServiceClient;
14
15  «visibility» class «name.toFirstUpper()–» {
16
17      public static void main(String[] args1) throws AxisFault {
18
19          RPCServiceClient serviceClient = new RPCServiceClient();
20          Options options = serviceClient.getOptions();
21          «FOREACH ((uml::Model) this.eRootContainer).eAllContents.
22          typeSelect(UP4WS::WebService) AS ws–»
23          EndpointReference targetEPR = new EndpointReference
24          ("http://localhost:8080/axis2/services/«ws.name»");
25          «ENDFOREACH–»

```

```

26
27     options.setTo(targetEPR);
28     «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
29         (UP4WS::ProxyImplementation) AS c-»
30         «c.name.toFirstUpper()-» client = new «c.name.toFirstUpper()-»();
31     «ENDFOREACH-»
32     «EXPAND createException
33     FOREACH eAllContents.typeSelect(UP4WS::ClientMain)-»
34     }
35 }
36 «ENDFILE»
37 «ENDDEFINE»

```

Listing 3.21: createClientImpl Template

3.3.1.5.3 createException Template for «ClientMain»

The createException template (Listing 3.22) generates the exception handler inside the main method. The exception handler invokes the methods that represent the provided services of the Web service. The real implementation of the exception handler is represented by assigning a state machine that applies the stereotype «OperationStateMachine» to the main method (Section 3.2.1.3).

```

1 «DEFINE createException FOR UP4WS::ClientMain»
2
3     try{
4         «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
5             (UP4WS::OperationStateMachine) AS wssm-»
6             «IF name+"In"+owner.name == wssm.name-»
7                 «EXPAND CommonTemplates::implementState(wssm) FOREACH
8                     wssm.eAllContents.typeSelect(UP4WS::OperationState)-»
9                 «ENDIF»
10            «ENDFOREACH-»
11    }catch (Exception e){
12        System.out.println(e);
13    }
14 «ENDDEFINE»

```

Listing 3.22: createException Template

3.3.2 Implementation Environment

For the validation of the profile implementation, different environments need to be specified and established. The implementation environment comprises all environments used for performing all the tasks related to this work, i.e. modelling, model transformation and code generation, and Web service execution. The following is a description of the different involved environments.

3.3.2.1 Modelling Environment

For the development of the UML profile and the UML model, a corresponding UML tool is needed. The UML tool should be able to export the profile and the model in the correct format.

3.3.2.2 Generator Environment

The generator environment can be set by generating a new Xpand project based on the instructions available at [XPA]. Part of the project specifications is to define the Modeling Workflow Engine (MWE) file. The MWE file is an XML document that specify all characteristics of the project, such as:

- **Input Model:** on which model the transformation will be executed.
- **Output Files/Outlets:** where the output will be located.
- **Cleaner Component:** to clean the output files at each run of the generator.
- **Generator Component:** to generate the output and direct them to the output files. More than one generator component can be specified in the same workflow file.

It is useful to define an external property file in order to define some variable data (name and location of the model, name of the profile, and so on). The advantage of the property file is to keep changes in the MWE file at their minimum, and enables reuse in different projects.

3.3.2.3 Web Services Execution Environment

In order to enable the execution of Web services, some environment characteristics have to be specified. The environment in this case has different variables, such as:

- The Web service engine (Apache Axis2).
- The application server (Apache Tomcat).
- The operating system (Windows).

3.3.2.4 Platform Configuration

The configuration of the platform in the profile implementation refers to the generation of the platform configuration files needed for the implementation and deployment of Web service. The files in consideration here are the *build.xml*, and the *services.xml* files. The *build.xml* file is responsible for the compilation of the Java files at the service provider side with *ant* [ANT], while the *services.xml* file is used for the deployment of the Web service in the application server. For both files, one template file (i.e. XmlFiles Template File) is developed. The XmlFiles template file contains the following templates:

- xmlRoot template for `uml::Model`

- createBuildXML template for «WebService»
- createServicesXML template for «WebService»

3.3.2.4.1 xmlRoot Template for uml::Model

The xmlRoot template (Listing 3.23) is used to invoke the *createBuildXML* and *createServicesXML* templates inside the same template file, and serves for reusability. The xmlRoot template must be configured inside the MWE file.

```

1 «DEFINE xmlRoot FOR uml::Model»
2   «EXPAND createBuildXML
3     FOREACH this.eAllContents.typeSelect(UP4WS::WebService)»
4
5   «EXPAND createServicesXML
6     FOREACH this.eAllContents.typeSelect(UP4WS::WebService)»
7 «ENDDEFINE»

```

Listing 3.23: xmlRoot Template

3.3.2.4.2 createBuildXML Template for «WebService»

The createBuildXML template (Listing 3.24) generates the *build.xml* file, which is used to compile the Java files at the server side and generate the archive file for the Web service (*serviceName.aar*). The *serviceName.aar* file will be located inside the services folder in the Apache Tomcat to host the Web service. The compilation is done using the *ant* command.

```

1 «DEFINE createBuildXML FOR UP4WS::WebService»
2   «FILE "build.xml" buildXML-»
3   <project name="«name»"
4     basedir="."
5     default="generate.service">
6     <property name="service.name" value="«name»"/>
7     <property name="dest.dir" value="build"/>
8     <property name="dest.dir.classes" value="${dest.dir}/${service.name}"/>
9     <property name="dest.dir.lib" value="${dest.dir}/lib"/>
10    <property name="axis2.home" value="../.."/>
11    <property name="repository.path" value="${axis2.home}/repository"/>
12
13    <path id="build.class.path">
14      <fileset dir="${axis2.home}/lib">
15        <include name="*.jar"/>
16      </fileset>
17    </path>
18
19    <path id="client.class.path">
20      <fileset dir="${axis2.home}/lib">
21        <include name="*.jar"/>
22      </fileset>
23      <fileset dir="${dest.dir.lib}">
24        <include name="*.jar"/>
25      </fileset>
26    </path>
27
28    <target name="clean">

```

```

29 <delete dir="${dest.dir}"/>
30 <delete dir="src" includes="sample/webservice/stub/**"/>
31 </target>
32
33 <target name="prepare">
34 <mkdir dir="${dest.dir}"/>
35 <mkdir dir="${dest.dir}/lib"/>
36 <mkdir dir="${dest.dir.classes}"/>
37 <mkdir dir="${dest.dir.classes}/META-INF"/>
38 </target>
39
40 <target depends="clean,prepare" name="generate.service">
41 <copy file="src/META-INF/services.xml" overwrite="true" tofile=
42 "${dest.dir.classes}/META-INF/services.xml"/>
43 <javac destdir="${dest.dir.classes}" includes="
44 sample/webservice/service/**,sample/webservice/data/**" srcdir="src">
45 <classpath refid="build.class.path"/>
46 </javac>
47 <jar basedir="${dest.dir.classes}"
48 destfile="${dest.dir}/${service.name}.aar"/>
49
50 <copy file="${dest.dir}/${service.name}.aar" overwrite="true"
51 tofile="${repository.path}/services/${service.name}.aar"/>
52
53 </target>
54
55 <target depends="clean,prepare" name="rpc.client">
56 <antcall target="rpc.client.compile"/>
57 <antcall target="rpc.client.jar"/>
58 <antcall target="rpc.client.run"/>
59 </target>
60
61 <target name="rpc.client.compile">
62 <javac destdir="${dest.dir.classes}" includes="
63 sample/webservice/client/**,sample/webservice/data/**" srcdir="src">
64 <classpath refid="build.class.path"/>
65 </javac>
66 </target>
67
68 <target name="rpc.client.jar">
69 <jar basedir="${dest.dir.classes}" destfile="${dest.dir.lib}/rpc-client.jar"
70 includes="sample/webservice/client/**,sample/webservice/data/**"/>
71 </target>
72
73 <target name="rpc.client.run">
74 <java classname="sample.webservice.client.«name»Client">
75 <classpath refid="client.class.path"/>
76 </java>
77 <java classname="sample.webservice.client.«name»ClientMain">
78 <classpath refid="client.class.path"/>
79 </java>
80 </target>
81 </project>
82 «ENDFILE»
83 «ENDDDEFINE»

```

Listing 3.24: createBuildXML Template

3.3.2.4.3 createServicesXML Template for «WebService»

The createServicesXML template (Listing 3.25) generates the *services.xml* file, which is required for the deployment of the Web service on the application server hosting the Web service.

```

1 «DEFINE createServicesXML FOR UP4WS::WebService»
2
3 «FILE "services.xml" servicesXML-»
4
5 «REM»get the name of the Web service«ENDREM»
6
7 <service name="«name" scope="application">
8
9 «REM»get the name of the comment attached to the Web service Java class «ENDREM»
10
11 <description>
12 «ownedComment.get(0).body»
13 </description>
14
15 <messageReceivers>
16 <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
17 class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
18
19 <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
20 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
21 </messageReceivers>
22
23 «REM»get the name of the Web service«ENDREM»
24 <parameter name="ServiceClass">sample.webservice.service.«name»
25 </parameter>
26 </service>
27 «ENDFILE»
28 «ENDDEFINE»

```

Listing 3.25: createServicesXML Template

3.3.2.5 Web Service Execution

The execution of the Web service means running the Web service on the selected platform. After setting and establishing the environments for the Web service, it should be possible to run and test the execution of the Web service. As part of the output files, a *README.txt* file is generated to instruct the user how to run the Web service. The *README.txt* file contains different types of information, which should help in understanding the entire process for executing the Web service. It illustrates mainly the following points.

- A short overview of the Web service application.
- Prerequisites for running the Web service on the platform.
- Compiling the Web service classes at the service provider side.
- Deploying the Web service on the application server.

- Compiling the Web service classes at the service client side.
- Running the Web service by executing the objects at the client side, which contain the main method.
- Further information for help and support.

Listing 3.26 represents the template, which is responsible for generating the *README.txt* file with the instructions for the execution of the Web service.

```

1  «IMPORT UP4WS»
2
3  «DEFINE readmeTplm FOR UP4WS::WebService»
4
5  «FILE "README.txt"–»
6
7  Introduction
8  =====
9  The («name» Web Service) has been implemented to validate the proposed
10 UML profile for Web services and the code generation process.
11 The case study shows a complete working example that represents the modelling of Web services
12 with UML, and the model transformation and code generation of source code and configuration
13 files for the relevant platform.
14 In the «name» Web service, both perspectives of service provider and service client have been
15 introduced.
16
17 Web Services using Apache Axis2
18 =====
19 The program contains the source code for the «name» Web service.
20 The source code is generated from a UML model developed using a UML Tool,
21 and transformed into a java code using the Xpand language together with Xtend
22 functions in Eclipse.
23
24 Prerequisites and Environment Settings
25 =====
26 1. Windows as an Operating System (The case study has been implemented on
27 Windows XP and Vista)
28 2. Apache Ant 1.6.2 or later
29 3. Apache Axis2
30 4. Apache Tomcat
31 5. Java
32
33 How to deploy the service on the Application Server?
34 =====
35 1. Compile the server side (service provider side) classes with ANT.
36 2. Put the «name».aar inside the services folder in Tomcat
37 (...apache-tomcat\webapps\axis2\WEB-INF\services)
38 3. Initiate the tomcat server by typing the relevant command, i.e. tomcat.exe or startup.bat
39 from inside the bin folder.
40 4. Make sure that the service is correctly deployed by typing
41 http://localhost:8080/axis2/services/«name»?wsdl in your browser
42
43 Compile the Client side Classes by typing the following commands:
44 =====
45 First go to the location, where the build.xml file resides:
46
47 javac –sourcepath src\sample\webservice\client –classpath build\«name»
48 –extdirs "C:\ApacheInstallation\axis2-1.5\lib"

```

```

49 -d build\«name» src\sample\webservice\client\«name»Client.java
50
51 javac -sourcepath src\sample\webservice\client -classpath build\«name»
52 -extdirs "C:\ApacheInstallation\axis2-1.5\lib"
53 -d build\«name» src\sample\webservice\rpcclient\«name»ClientMain.java
54
55 Running the Client Side classes by typing the following commands:
56 =====
57 First go to the location which ends with ...\build\«name»
58 N.B. The name of the folder that contains your Web service files
59 must hold the same name of your Web service.
60
61 java -Djava.ext.dirs="C:\ApacheInstallation\axis2-1.5\lib"
62 sample.webservice.client.«name»ClientMain
63
64 Help
65 ===
66 Please contact (wdahman@informatik.uni-goettingen.de or arrivalsw@yahoo.com)
67 if you have any trouble running the Web service.
68
69 «ENDFILE»
70 «ENDDEFINE»

```

Listing 3.26: README Template

3.4 Summary

This chapter has introduced a general **UML** profile for Web services. The **UP4WS** aims at enabling the automatic generation of executable Web services by defining a set of **UML** extensions for Web services. These extensions are represented by means of stereotypes, where each stereotyped element can participate in the model transformation and code generation process. The latter has been introduced as part of the profile implementation. For the code generation process, Xpand transformation rules are defined for transforming the **UML** profile and the **UML** model into source code and configuration files for the execution of Web services. The Web service is supposed to run on the Apache Axis2/Java Web service engine, which has been selected as a platform for its execution. In order to run the Web service on another platform, the Xpand transformation rules need some modifications, especially at the service client side, where platform specific artifacts can be found. However, the basic extensions in the **UP4WS** are sufficient for Web services, since they represent all main elements for Web service disregarding the type and nature of the execution platform.

Chapter 4

Web Services Development Model

The Web Services Development Model (**WSDM**) specifies a set of activities that aim at the development of Web services with **UML**. It is defined in terms of tasks, which in turn define their own sub-tasks. The **WSDM** defines mainly three tasks for the development of Web services. The first is the requirements analysis task, which specifies what the Web service should offer. The second is the design task, which specifies how to implement the Web service by providing a model that reflects the specifications of the Web service. The third is the implementation task, where the Web service specifications are transformed into a complete and executable source code and configuration files. The use of the UML Profile for Web Services (**UP4WS**) within the **WSDM** is mandatory. The model elements that will apply the **UP4WS** stereotypes must be specified. Figure 4.1 presents

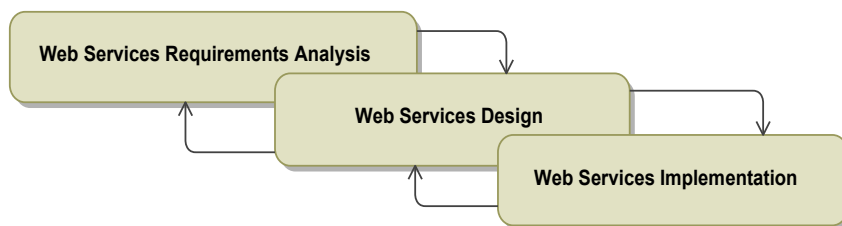


Figure 4.1: Web Services Development Model Overview

an overview of the main tasks of the **WSDM**. The figure shows intersections between the tasks to emphasise that some activities can be done in parallel and not necessarily in a sequential manner.

4.1 Requirements Analysis

This task includes capturing the requirements, identifying the stakeholders, and building the initial architecture of the Web service application. This task is divided into two sub-tasks, i.e. requirements elicitation and requirements specification and modelling. For each sub-task, a description of the task and how UML fits into it will be presented.

4.1.1 Requirements Elicitation

In the requirements elicitation sub-task, the requirements of the Web service application are to be identified and gathered. The requirements at this step mean identifying what services the Web service will provide. For example, a statement like *"The Web service shall calculate the sum of two numbers"* can represent one of the services that a Web service shall provide. Such a requirement is known as a *functional requirement* for the Web service. Furthermore, quality attributes can be considered as requirements, for example, availability, and reliability are types of possible requirements. Such requirements are known as *non-functional requirements*. The requirements will be collected and captured from different resources such as the users of the Web service, the domain, and current and historical data of similar Web services.

It is important to define a general function for the Web service, which represents a collective or common goal for the Web service. The function will then be decomposed into smaller functions representing the exact services provided by the Web service. The domain of the Web service application must be examined to identify special requirements imposed by the domain, as well as the constraints of business, technology, laws. The customer will play an important role in achieving this purpose. Different techniques, such as interviews and questionnaires can take place in identifying and gathering the requirements for the Web service.

4.1.2 Requirements Specification and Modelling

During and after the requirements elicitation, the specification and modelling of requirements shall take place. The modelling can be done by drawing some figures and shapes reflecting the requirements in understandable way to the customer. The proposed WSDM uses UML as a modelling notation for Web services. The allocation of UML to the tasks and sub-tasks will be based on the best practises proposed by the RUP.

4.1.2.1 UML Support

UML provides different types of diagrams that can be used to capture requirements. This thesis proposes two types of UML diagrams for this purpose, i.e. the use case diagram, and the class diagram.

4.1.2.1.1 Use case diagram

The initial step to model the Web service is to develop the use case diagram. This will initially specify the boundaries of the Web service application. It also shows the services of the Web service in terms of use cases, since each one can represent one of these services. Any use case can extend other use cases or include them. The actors in the use case diagram represent the objects outside the Web service application that exchange data with the Web service and interact with it. They either send data to the Web service application, or receive data from it, or both. The use case diagram shall be modelled in two steps, where the first step is to represent the Web service in only one use case. After that, the single use case shall be decomposed into more use cases according to the number of services in the Web service. Several steps of decomposition of use cases on multiple levels can occur. Figure 4.2 shows an example of use case diagrams for a weather Web service. The first use case diagram in Figure 4.2(A) shows one use case that represents a collective goal of the Web service. It also shows three actors, two of which represent normal client applications *Client A* and *Client B*, in addition to another Web service (*Web Service X*), which can also interact with the Web service. The second use case diagram in Figure 4.2(B) extends and refines the first one (Figure 4.2(A)). It decomposes the use case in Figure 4.2(A) into two use cases representing the exact services of the Web service. Further refinement of the use cases in other use case diagrams shall be possible. Actors remain the same, although they can be refined into more detailed actors. Each actor can request one or more of the provided services. It also shows the system boundaries by means of a rectangular shape, which distinguishes between the internal and external parts involved in the Web service.

4.1.2.1.2 Class Diagram

Class diagrams are very important in any UML model. They identify the requirements of Web services, define its architecture, and specify how each element serves in achieving the final goal of the Web service. In addition, they represent the objects in the Web service, such as persons, systems, units, objects and how they are associated. Class diagrams shall be used to represent the basic UML extensions for Web services proposed in the UP4WS.

Several techniques can be adopted to identify the objects in the class diagram. One of the techniques is the commonly used Class, Responsibilities, Collaborations

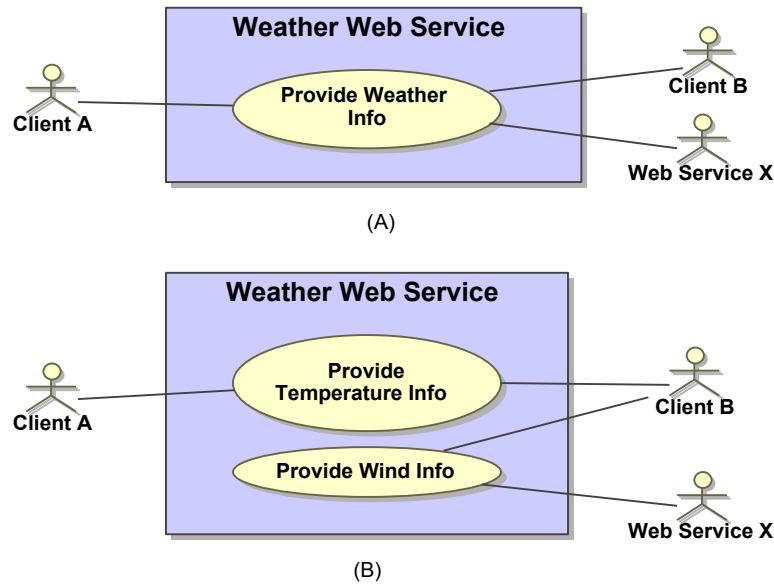


Figure 4.2: Sample Use Case Diagrams

Service Object (Weather Service)	
Responsibilities	Collaborations
set day temperature provide day temperature set night temperature provide night temperature provide average of the temperature set direction of the wind provide direction of the wind	data object (Temperature and Wind) proxy client object (Weather proxy client)

Table 4.1: CRC Card for Service Object

(CRC) analysis. In the CRC analysis, nouns and verbs in the description are analysed. Each noun is a candidate to be an object, while each verb is a candidate to be an operation of an object. The operations represent the responsibilities of the objects, while the associations with other objects reflect the collaborations between the objects. The collaborations represent the possible associations of the object with other objects. Table 4.1 shows a sample CRC card for the service object (e.g. *Weather Service*). The card is composed of two columns, the responsibilities column shows the functionalities of the *Weather service* such as *set day temperature* and *provide day temperature*. The other column reflects the collaborations, which represents the associations with other objects, such as *Temperature*, *Wind* and *Weather proxy client*. The CRC cards can be done for each candidate object in the Web service to capture its complete specifications.

The class diagrams in the requirements analysis task are drawn in an abstract way and display only the objects in the Web service and how they relate to each other by means of associations, without going into details. The relevant details will be added during the design task. Figure 4.3 shows an initial architecture for a sample weather Web service. The figure shows the objects in the service provider side, and the service client side.

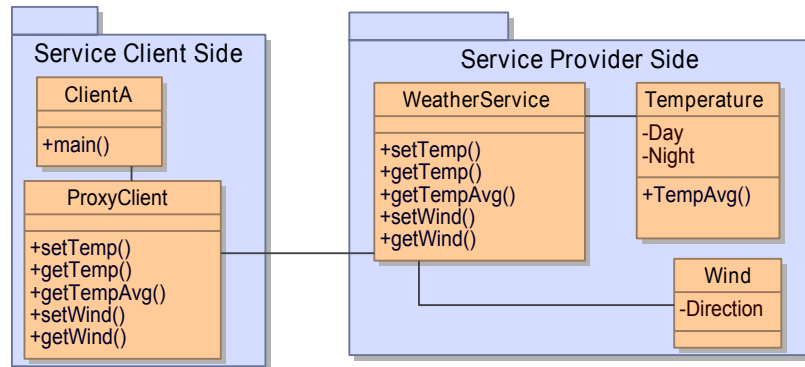


Figure 4.3: Class Diagram for Requirements Analysis

4.2 Web Service Design

The Web service design is the task of realising the requirements gathered in the requirements analysis task. The realisation means that the output of the requirements analysis task will be extended and refined to reflect the complete specifications of the Web service, in a concrete and detailed way, in order to enable the implementation of the Web service. The design task aims at building the Web service architecture and identifying the behaviour of the Web service. The **WSDM** proposes two steps to design the Web service, i.e. realisation of architecture design, and realisation of behaviour design. Both steps can be done in parallel.

4.2.1 Realisation of Architecture Design

The architecture design is concerned with developing the Web service architectures using class diagrams and adding the concrete details of the Web service. These details shall represent the complete architectural specifications of the Web service. The design of Web service architecture must take the target platform into consideration. For example, if the Web service is implemented in the Java language, no multiple inheritance shall be allowed in the **UML** model. The syntax of the declarations of classes, attributes, operations imposed by Java must be

followed. This is also applicable to the platform, since it may constrain the use of some constructs or artifacts.

4.2.1.1 UML Support

To build the Web service architecture, the **UML** class diagrams shall be used. They have been initially proposed in the requirements analysis task for requirements elicitation, and they will also be used for the architecture design.

4.2.1.1.1 Class Diagrams

The class diagrams shall be more concrete and reflect the whole Web service specifications at this step. Such specifications are related to the requirements themselves, and to the target platform. Different types of refinements shall be considered in the class diagram such as:

- **Associations:** e.g. the exact type of the association between the objects, multiplicity, directions.
- **Attributes:** e.g. the exact declaration of the attributes, type, visibility, default value, initialisation.
- **Operations:** e.g. the exact and full signature of each operation, visibility, return type, parameters and their types.

All the above points are samples of the details that should be reflected in the class diagrams in the design task.

4.2.1.2 Identifying UML Extensions for the Web Service

At this step, the basic **UML** extensions for Web services in the **UP4WS** must be specified. The development team must specify the model elements that will apply these extensions from the requirements gathered in the requirements elicitation task. From the description of the Web service, it should be possible to capture the basic extensions of the Web service. In Web services, it is important to notice that clients who request the service are not part of the Web service itself, since they represent external applications. However, the Web service can request each other. In this case, the Web service plays the role of service provider and service client at the same time. This is why clients are not considered parts of the Web service itself, but parts of the entire Web service application. The following steps shall help in identifying the basic **UML** extensions for the Web service:

1. Distinguish between the service client side and the service provider side.
2. In the service provider side, identify the object that will represent the service. This will be known as the *service object*. The services will be represented as methods in the service object.

3. Specify the data required by the service object. These *data objects* must be identified together with their *data variables*.
4. At the client side, the *proxy client object* that implements the services from the client point of view shall be specified.
5. The *client object*, which includes the execution mechanism shall be specified.

From Figure 4.3, which represents a class diagram that is developed in the requirement analysis task, the following UP4WS elements shall be allocated on those objects and their own elements according to the description in Section 3.1.

- *WeatherService* represents the service object.
- The methods *setTemp*, *getTemp*, *getTempAvg*, *setWind*, and *getWind* in the *WeatherService* and *ProxyClient* objects represent the services.
- *Temperature* and *Wind* represent the data objects.
- The *Day* and *Night* attributes in *Temperature* object, and *Direction* attribute in the *Wind* object represent the data variables.
- The *TempAvg* method represents a method that adds behaviour to the owning object. The behaviour of the owning object shall be represented in a state machine diagram according to the specifications of the UP4WS.
- *ProxyClient* represents the proxy client object that implements the services from the client's perspective.
- *ClientA* represents the client object that contains the execution mechanism.
- The *main* method represents the execution method in the implementation programming language.

4.2.1.3 Allocating UML Extensions to the Design Architecture

Based on the previous section, where the identification of the candidate extensions took place, the UML basic extensions for Web services shall be allocated to the identified objects and the attributes and operations in each object. The UP4WS has defined a specific set of stereotypes that must be found in any Web service application as follows:

- «WebService» for the service object.
- «ProxyMethod» for the operations that represent the services.
- «DataContainer» for the data objects.

- «DataElement» for the attributes that represent the data variables.
- «ProxyImplementation» for the proxy client object.
- «Client» for the client object that contains the execution mechanism.
- «ClientMain» for the main method in the client object.

Figure 4.4 shows two class diagrams representing the main objects in the service provider side in Figure 4.4(A), and the service client side 4.4(B) for a weather Web service. Compared to the class diagrams in Figure 4.3, the class diagrams reflect additional information about the object, such as the type of the attributes and the return type and parameters of the operations. However, it could include even more information, such as association types, role names, and so on. The class diagrams also present the allocation of the stereotypes defined by the UP4WS. They appear on the relevant elements in the UML model representing the Web service.

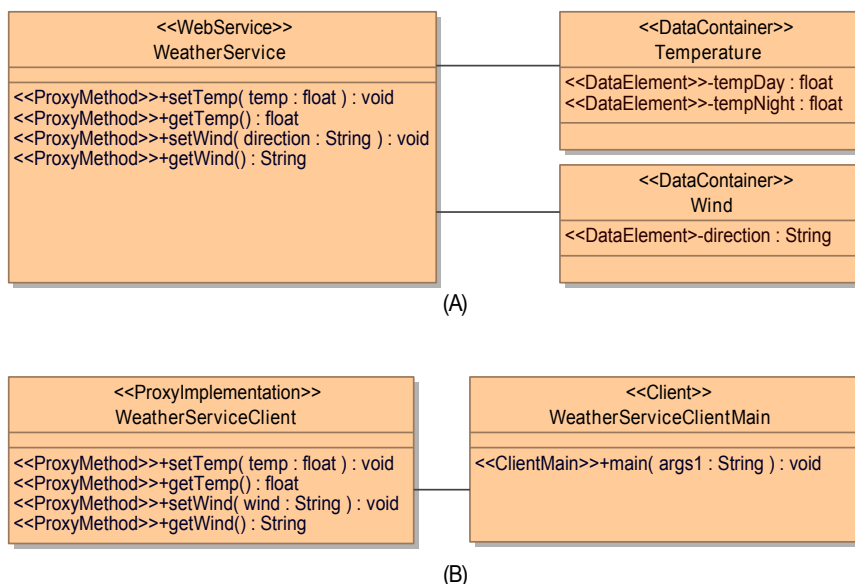


Figure 4.4: A Class Diagram in Architecture Design with UP4WS Stereotypes

4.2.2 Realisation of Behaviour Design

The realisation of the behaviour design is concerned with adding the relevant behaviour to the objects. Operations specify some behaviour of the owning object, since invoking an operation might change some values of the object, or cause a change in the object's state. The WSDM specifies two types of dynamic behaviour, i.e. the behaviour, which corresponds to the state of an object, and the

behaviour, which corresponds to the values of an object. For the former, a state machine diagram shall be assigned to the object that has the behaviour. The state machine shall apply the stereotype «ObjectStateMachine». For the latter, a state machine applying the stereotype «OperationStateMachine» shall be assigned to the operation of an object. However, the behaviour is still owned by the object owning that operation, since operations themselves do not have owned behaviour.

4.2.2.1 UML Support

UML provides different types of diagrams to represent the dynamic behaviour of objects such as activity diagram, and state machine diagram. The WSDM suggests using the state machine diagram for the representation of Web service behaviour.

4.2.2.1.1 State Machine Diagrams

State machines are powerful means to model the dynamic behaviour of software systems. In addition, many UML tools provide techniques to generate a source code from UML state machines. State machines enable specifying the behaviour of any classifier, including classes, sub-systems, and the whole system. Any state machine can represent the dynamic behaviour that describes a change in the object's state. In this case, the transition from one state to another takes place in response to a specific trigger or event, which normally corresponds to an operation call. The UP4WS specified two extensions for state machines that can be used for the realisation of the Web service behaviour as described in Section 3.2.1.

4.2.3 Designing the Web Services Platform

The target at this step is the selection of the most relevant platform for the implementation and deployment of the Web service under development. Web service are targeting the enhancement of interoperability between distributed systems. Therefore, they have unique platform specifications. The development team shall decide on which platform the Web service will be deployed and implemented. The design of the Web services platform includes a set of activities in order to find out the most relevant platform for the Web service. These activities are summarised in the following.

- **Setting the selection criteria:** in order to reach the suitable decision on the platform, it is very important to specify the criteria for such a decision. Several platforms can be used to deploy and implement Web services. Therefore, the decision should be justified. The platform for Web services has architecture that differs from typical software that runs on a single loca-

tion, and thus the selection has some novelty. The following are some points that should be considered when selecting the platform for Web services.

- **Compatibility:** In most cases, Web services extend normal application and enable them to work over the Internet or other type of network. Therefore; it is necessary for the platform to be compatible with the existing technology used for the current application, or in need for slight changes that would not result in radical modifications in the current infrastructure. The platform should also be compatible with the abilities of the team developing the Web service. For example, if the team members are familiar with a specific programming language, the platform is supposed to enable the implementation of the Web service in the same programming language.
- **Cost:** this includes the costs resulting from the adoption of certain platform. It could include different types of the costs such as:
 - * **License Costs:** if the license for platform has to be purchased.
 - * **Hardware Costs:** if the platform requires additional hardware.
 - * **Training and Material:** if the users need training and supporting material.

It is important to prioritise the criteria according to the exact needs. This will make the selection decision easier and justifiable.

- **Investigating different platforms:** the team should survey the market in order to find out the relevant Web service platform. In this case a list of the existing platforms should be provided together with the corresponding information. The team classifies the investigated platforms according to the specified selection criteria.
- **Selecting the relevant platform:** after investigating the different platforms, the selection of the relevant platform should be made. A full documentation of the selection process as well as the platform should be prepared in order to enable the implementation and the code generation process.

4.3 Web Service Implementation

The implementation task clarifies how the Web service will be implemented, and how the model will be transformed into the target output. This task receives the detailed design specifications from the design task, i.e. *Web service design*, in order to build and develop complete implementation specifications. At this step, the team should decide the parameters for the code generation process and prepare the plan for it. The plan should indicate several important points as follows:

- **Input model:** What are the model elements that will be used for the code generation process? Which **UML** elements will be used in the model transformation and code generation process?
- **Programming language:** to which programming language the model elements will be transformed? This is important, since some pieces of the code might be inserted manually in the input model.
- **Code generation environment:** This includes the selection of the code generator (*code generation engine*), and the transformation language, which works on it. Similar to the programming language for Web service implementation, the team should be familiar with the transformation language used in the code generation process.
- **Implementation environment:** this comprises all aspects of the implementation environment and considers its specifications. The team should identify the requirements for the implementation platform, since they influence the type of output files that shall be generated. The implementation environment includes the operating system, the Web service engine, the type of network.
- **Source code and configuration files:** the team should specify the source code and all other configuration and documentation files that should be generated.

4.4 Summary

This chapter has presented a model for the development of Web services. The development model is divided into tasks, where each one adds value to the development process. **UML** has been adopted as a modelling notation in the proposed **WSDM**. The **WSDM** constrains the use of the **UP4WS** in the relevant tasks. This is important, since the model transformation and code generation process will handle the stereotyped elements in the source model. The **WSDM** utilised different types of **UML** diagrams such as use case and class diagrams for the requirements analysis. Class diagrams have also been used together with state machine diagrams for the design and implementation of Web services. The use of the UML Profile for Web Services (**UP4WS**) is obligatory in the **WSDM** to enable the code generation process by identifying the relevant **UML** extensions for Web services.

Chapter 5

Case Study: Library System Web Service

For the validation of the **UML** profile for Web services and the code generation process, a case study has been implemented in order to prove the feasibility of the approach introduced in this thesis. The selected case study has been implemented in the following sequence of steps and represented in Figure 5.1:

- Modelling the requirements and specifications of the selected case study by the Magic Draw tool [MD].
- Exporting the model and the profile in an **EMF UML2 XMI** format to be used by the generator for code generation.
- The generator includes the definition of the transformation rules in Xpand according to the profile implementation illustrated in Section 3.3.
- After running the generator, the source code and the associated configuration files will be used together for the execution of the library system Web service. This includes setting the environment for the implementation.

5.1 Service Description

The library system Web service has been chosen for this thesis because of its well-known characteristics and the familiarity of its functionalities. The functionalities of the library system Web service are almost the same for any similar application and thus understandable by the audience. Each functionality represents one of the services provided by the library system Web service.

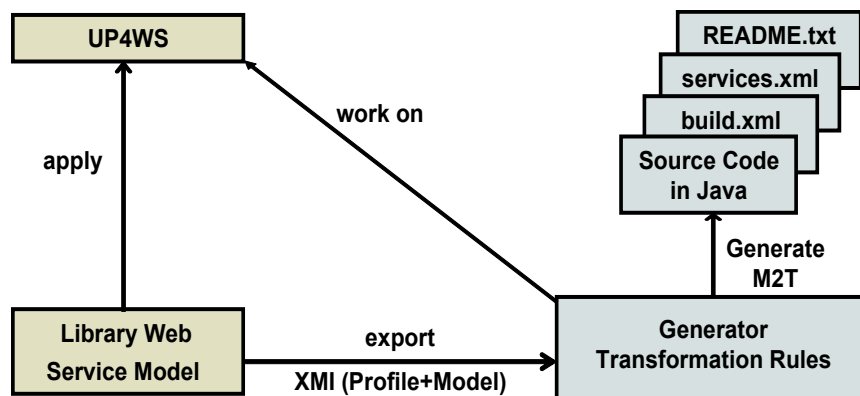


Figure 5.1: Case Study Implementation

A typical library system could have the following brief description:

The library system Web service manages the virtual book list of the library. Each book is distinguished via title, ISBN, and author. The library system Web service provides mechanisms for different processes on books; mainly, the client shall be able to add books to the book list. The books become available and can be borrowed. If a book is borrowed, it becomes on loan. The borrowed book shall be returned to the book list and becomes available again.

5.2 Library System Web Service Analysis

In the requirements analysis task, the guidelines and techniques proposed by the WSDM to gather and analyse the requirements are followed. In the requirements analysis task, the following set of activities take place:

- Specifying the main goal of the library system Web service, and its sub-goals as well,
- Specifying the main objects in the Web service,
- For each object, specify its responsibilities and collaborations. This is done by using the CRC cards technique.
- Modelling the requirements using the use case and class diagrams according to the guidelines of the WSDM from Chapter 4.
- Identify the main UML extension for the Web service according to the specifications of the UML Profile for Web Services (UP4WS) from Chapter 3.

Library System Service Object	
Responsibilities	Collaborations
provide adding books service	Book
provide lending books service	Book list
provide returning books service	Client

Table 5.1: CRC Card for Library System Service Object

From the library system Web service system description, the following objects are identified. Normally each name that appears in the description is a candidate to be an object within the Web service.

- *Library System service* (service object).
- *Book*, and *Book List* (data objects).
- *Client* (client object).

This is the initial analysis for the library system Web service description, which aims at identifying the objects in the library system Web service. Further analysis for the identified objects can be done by means of CRC cards. The WSDM proposes the CRC cards technique to analyse the objects in order to capture their responsibilities and collaborations. As an example for the usage of the CRC cards in the requirements analysis, the CRC cards of the objects *Library System Service* and *Book* objects are presented in the following:

- **Library System Service Object:** The *library system service* object represents the object that includes the services provided by the library system Web services. The CRC card in Table 5.1 gives an overview of the responsibilities of the library system service object and its collaborations. From the CRC card in table 5.1 it is possible to identify the main services of the library system Web service. The following is a list of the identified services:
 - **Adding Books:** enables clients to add new books to the book list.
 - **Lending Books:** enables clients to borrow books from the book list.
 - **Returning Books:** enables clients to return books.

The use case diagram in Figure 5.2 summarises the main services of the library system Web service as use cases. As specified in the WSDM, the use case diagram is modelled in, at least, two steps. The first step is to represent the collective goal of the Web service as one use case as shown in Figure 5.2(A), the collective goal is to manage the books in the book list. The second step is to decompose the use case representing the collective

Book Object	
Responsibilities	Collaborations
set book title	Book List
get book title	Library System Service
set book ISBN	
get book ISBN	
set book author	
get book author	

Table 5.2: CRC Card for Book Object

goal into more use cases that represent all the services of the Web service. Each use case will correspond to one of the services. This is shown in Figure 5.2(B).

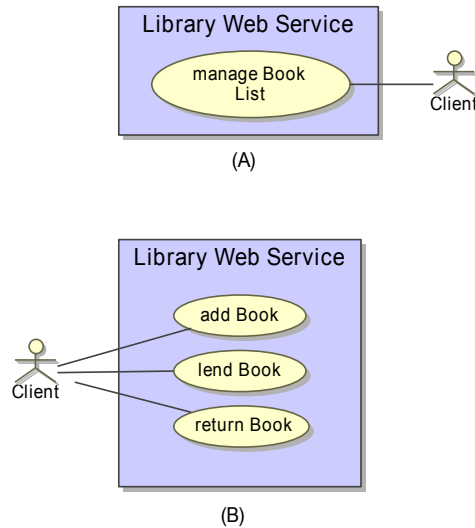


Figure 5.2: Requirements Analysis: Library System Web Service Use Cases

- Book Object:** The *Book* object is representing a data object. It is the main data object in the library system Web service. The *Book* object contains three data variables to distinguish it. Furthermore, the *Book* object has multiple states (borrowed or available). Therefore, it has a behaviour, which is represented by a state machine diagram as suggested by the WSDM. However, this will be handled in the design task, since the WSDM proposes the state machines for modelling the behaviour in the design task and not in the requirements analysis task. Figure 5.3 shows the initial architecture for the library system Web service. It identifies objects according to the library system Web service description. The objects at

the service provider side appears in Figure 5.2(A), while the objects at the service client side appears in Figure 5.2(B).

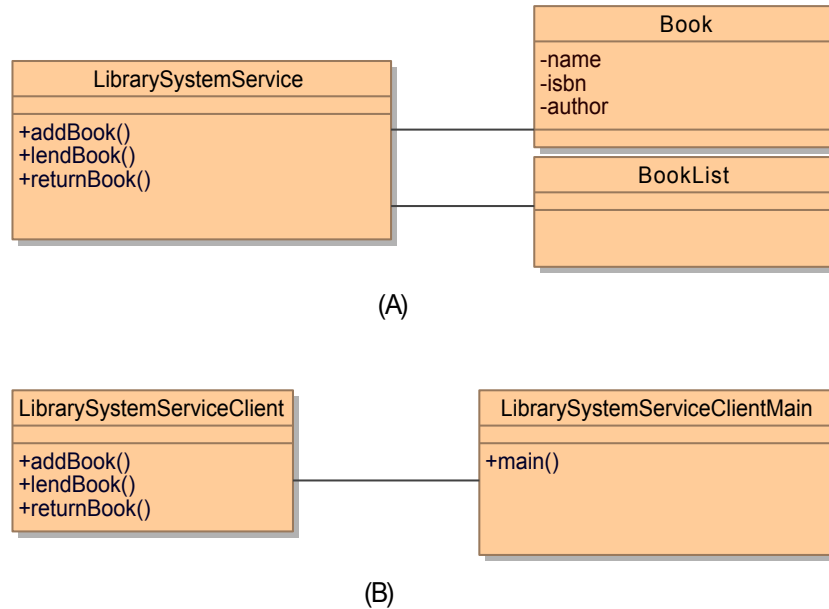


Figure 5.3: Library System Web Services Initial Architecture

5.3 Library System Web Service Design

In the design task, the findings of the requirement analysis task are refined and extended in order to enable the implementation of the library system Web service. At this step, the following activities take place:

- Allocate the **UML** extensions for Web service to the relevant elements in the **UML** model for the library system Web service.
- Refine and extend of the library system Web service architecture. This is done by extending the class diagram.
- Represent the behaviour of the data objects in the library system Web service.

5.3.1 Identifying and Allocating **UML** Extensions for the Library System Web Service

At this point, we identify the **UML** extensions for the library system Web service. This is the initial step for allocating the stereotypes on the elements that represent the **UML** extensions for Web services. **UP4WS** defined a set of basic extensions

that must be specified for any Web service application. The following is a list of the identified basic extensions for the library system Web service:

- *Library System Service* represents the service object.
- *Add Book*, *Lend Book*, and *Return Book* represent the provided services.
- *Book* and *Book List* represent the data objects.
- *Book* name, *ISBN*, and *author* represent data variables.
- *Client* represents client object.

Figure 5.3 in section 5.2 gave an overview of the main objects in the library system Web service, their attributes and operations, and their associations to each other. The UP4WS divides the client side objects into two objects, i.e. *proxy client object* to implement the services from the view point of the client, and *client* object that contains the execution mechanism for executing the Web service. The allocation of the UP4WS stereotypes on the UML model elements for the library system Web service is shown in Figure 5.4. The following are examples for the allocation of the stereotypes on the objects of the library system Web service and other elements in the model.

- *LibrarySystemService* extends the «WebService» stereotype.
- *Book* and *BookList* extend the «DataContainer» stereotype.
- *isbn*, *title*, *author* in *Book* class extend the «DataElement» stereotype.
- *getBook*, and *lendBook* in *LibrarySystemService* class extend the «ProxyMethod» stereotype.
- *getBookInfo* in *Book* class extends the «DataBehavior» stereotype.

It is important to notice that setter and getter methods are generated automatically in the source code according to the implementation specifications of the UP4WS. Therefore, they do not appear in the UML model. Instead, the «DataElement» is applied by the attributes that need to be accessed by setter and getter methods (see Sections 3.1.3 and 3.3.1.1.4).

5.3.2 Refinement of Library System Web Service Architecture

The refinement of the architecture of the library system Web service aims at adding all the details to the UML model, so it reflects the complete specifications of the library system Web service. At this step, the target platform and the programming language for the library system Web service implementation are taken into consideration, since they influence the type and nature of the added details.

For example, the programming language influences the representation of the book list object and the mechanism of storing the books. Since the library system Web service is implemented in the Java programming language, the HashMap mechanism from Java has been chosen to represent the book list as a table.

Figure 5.4 gives an overview of the library system Web service objects at the service provider side. It shows how the different types of details are added to the class diagram from the requirements analysis task. The figure reflects different details that have been recognised during the design of the library system Web service. For example, the method *getBook* has been added to the *LibrarySystemService* class, and the *BookList* class. Furthermore, the *Book* class includes additional operation *getBookInfo* to return a String of information about a specific book. In addition, the direct association between *LibrarySystemService* and *Book* object has been replaced with an indirect relation via the *BookList* object.

5.3.3 Representing Library System Web Service Behaviour

The description of the library system Web service indicates that the *Book* object has different states, i.e. *available*, and *borrowed*. For the representation of the *Book* object behaviour, a state machine diagram has been used as proposed by the WSDM. The state machine diagram enables the implementation of the behaviour of the corresponding object. Representing the behaviour of the *Book* object influences the class diagram in Figure 5.4, since it adds additional elements in the class diagram. For example, two enumeration classes have been added to represent the different states of the *Book* object (*BookStatusStates*), and the events that trigger the transitions between the states (*BookStatusEvents*). In addition, the *getBookStatus* operation has been added to the *LibrarySystemService* object to return the status of the *Book* object. Figure 5.5 represents the state machine diagram modelled to represent the behaviour of the *Book* object. The state machine diagram as specified in the UP4WS is named *BookStatus*. It contains three states, i.e. *NotAvailable*, *Available*, and *OnLoan*. The state *NotAvailable* is added to represent the default state of any book instance before it is added to the book list.

The *BookStatus* state machine diagram in Figure 5.5 applies the «ObjectStateMachine» stereotype in order to enable its implementation and the generation of the source code. The «ObjectState», «ObjectTransition», and «ObjectFinalState» stereotypes are parts of the implementation of the state machine that applies these «ObjectStateMachine» stereotype.

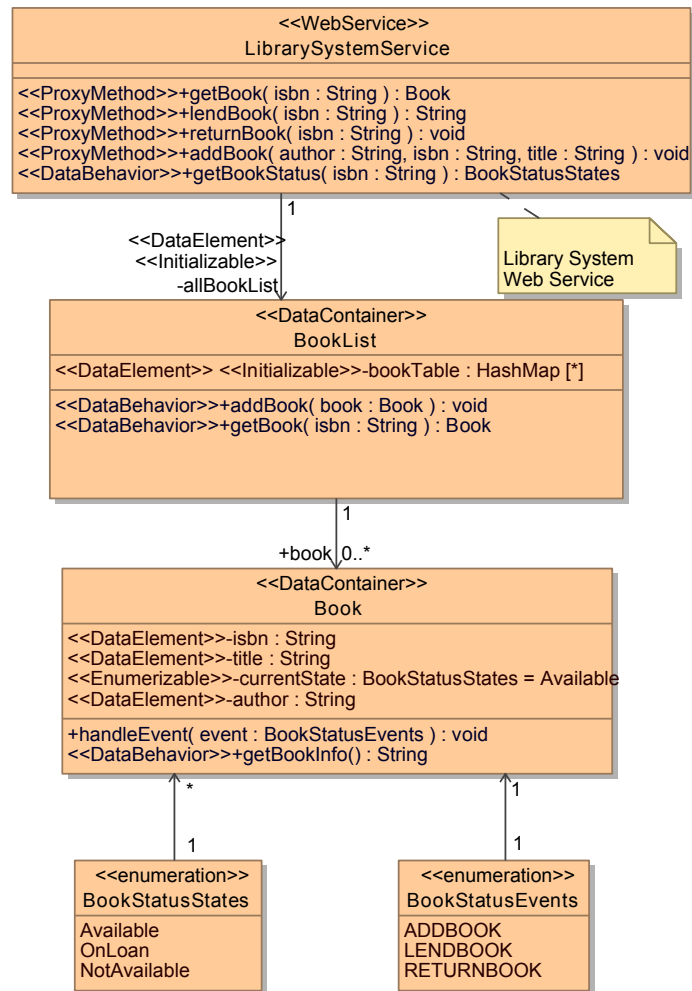


Figure 5.4: Class Diagram at Service Provider Side

5.4 Library System Web Service Implementation

The implementation of the library system Web service includes the model transformation process and the execution of the library system Web service on the selected platform. The implementation process produces the following output in order to enable the library system Web service execution:

- Source code in Java at service provider and service client sides,
- Configuration files for the Web service engine (*build.xml* and *service.xml* files), and
- Documentation file (*README.txt* file).

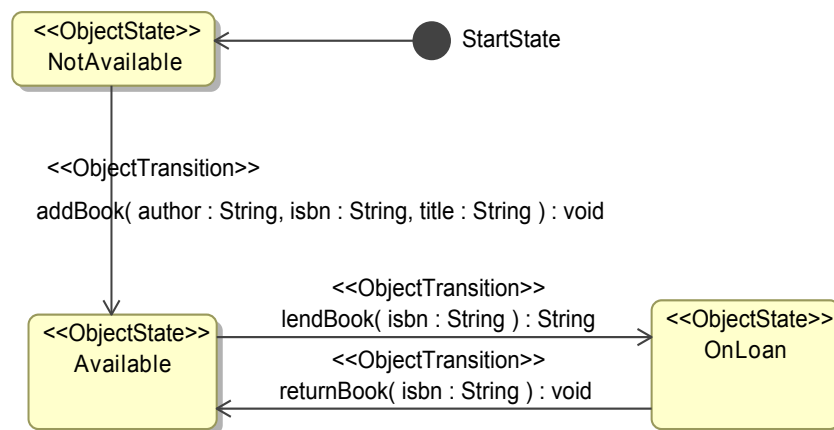


Figure 5.5: Book Status

5.4.1 Running the Generator

The model transformation process has been performed using the Xpand transformation language on the Eclipse platform. The generator receives the source UML model for the library system Web service as an Eclipse Modeling Framework (EMF) UML2 XML Metadata Interchange (XMI) file. This is the file format used in the library system Web service and exported using the Magic Draw UML tool. Other format types might need to be adapted in order to be read correctly by the Eclipse/Xpand generator. Running the Eclipse/Xpand generator requires the definition of the Modeling Workflow Engine (MWE) file. The MWE file specifies different parameters of the project, such as the source model, metamodel, and location of output files. Listing 5.1 represents the MWE file used to generate the output files for the library system Web service. The MWE file includes different types components, which serve for various purposes. The following list describes briefly the main components in the MWE file:

- Cleaner component to clean the output folder at each run of the generator.
- Generator component, which is responsible for generating the Java source code. It includes different outlets to specify the exact location for each Java file.
- Generator component for generating the configuration files and to specify, where each file should be located.

```

1 <?xml version="1.0" encoding="windows-1252"?>
2 <workflow>
3   <property file="workflow.properties"/>
4
5   <bean class="org.eclipse.xtend.typesystem.uml2.Setup"
6     standardUML2Setup="true"/>
  
```

```

7
8 <component class="org.eclipse.xtend.typesystem.emf.XmiReader">
9   <modelFile value="{modelFile}"/>
10  <outputSlot value="model"/>
11 </component>
12
13 <!-- Cleaner Component -->
14 <component id="dirCleaner"
15   class="org.eclipse.emf.mwe.utils.DirectoryCleaner" >
16   <directory value="{srcGenPath}"/>
17 </component>
18
19 <!-- Generator Component -->
20 <component id="generator" class="org.eclipse.xpand2.Generator"
21   skipOnErrors="true">
22   <metaModel
23     class="org.eclipse.xtend.typesystem.emf.EmfRegistryMetaModel"/>
24   <metaModel class="org.eclipse.xtend.typesystem.uml2.UML2MetaModel"/>
25
26   <metaModel id="ApplicationProfile"
27     class="org.eclipse.xtend.typesystem.uml2.profile.ProfileMetaModel">
28     <profile value="UP4WS.profile.uml"/>
29   </metaModel>
30
31
32   <expand
33     value="templates::main::root FOR model"/>
34     <fileEncoding value="ISO-8859-1"/>
35
36
37     <outlet path="{srcGenPath}">
38       <postprocessor class="org.eclipse.xpand2.output.JavaBeautifier"/>
39       <postprocessor class="org.eclipse.xpand2.output.XmlBeautifier"/>
40     </outlet>
41
42     <outlet name= 'myServiceOutlet'
43       path='LibrarySystemService/src/sample/webservice/service'>
44       <postprocessor class="org.eclipse.xpand2.output.JavaBeautifier"/>
45     </outlet>
46
47     <outlet name= 'myDataOutlet'
48       path='LibrarySystemService/src/sample/webservice/data'>
49       <postprocessor class="org.eclipse.xpand2.output.JavaBeautifier"/>
50     </outlet>
51
52     <outlet name= 'myClientOutlet'
53       path='LibrarySystemService/src/sample/webservice/client'>
54       <postprocessor class="org.eclipse.xpand2.output.JavaBeautifier"/>
55     </outlet>
56
57
58     <outlet name= 'myServicesXMLOutlet'
59       path='LibrarySystemService/servicesXMLFile'>
60       <postprocessor class="org.eclipse.xpand2.output.XmlBeautifier"/>
61     </outlet>
62
63     <outlet path="LibrarySystemService" append="true"
64       name="APPEND" overwrite="true"/>
65

```

```

66 <!-- Generator Component -->
67 <component id="generator" class="org.eclipse.xpand2.Generator"
68   skipOnErrors="true">
69   <metaModel
70     class="org.eclipse.xtend.typesystem.emf.EmfRegistryMetaModel"/>
71   <metaModel class="org.eclipse.xtend.typesystem.uml2.UML2MetaModel"/>
72
73   <metaModel id="profile"
74     class="org.eclipse.xtend.typesystem.uml2.profile.ProfileMetaModel">
75     <profile value="UP4WS.profile.uml"/>
76   </metaModel>
77
78   <expand
79     value="templates::XmlFiles::xmlRoot FOR model" />
80   <fileEncoding value="ISO-8859-1"/>
81
82   <outlet path='${MyXMLfiles}' append="false" name="APPEND"/>
83   <outlet path='${MyXMLfiles}' overwrite="true"/>
84
85   <outlet name= 'servicesXML' path='LibrarySystemService/src/META-INF'>
86     <postprocessor class="org.eclipse.xpand2.output.XmlBeautifier"/>
87   </outlet>
88
89   <outlet name= 'buildXML' path='LibrarySystemService'>
90     <postprocessor class="org.eclipse.xpand2.output.XmlBeautifier"/>
91   </outlet>
92
93 </component>
94
95 </workflow>

```

Listing 5.1: MWE File for the Library System Web Service

5.4.1.1 Generating Java Source Code

Since the *Book* object has a behaviour represented by a state machine diagram, it has been selected as a sample for the generation of the Java source code. The *Book* object applies the «DataContainer» stereotype and contains other elements, namely attributes and operations that apply different stereotypes. Furthermore, a state machine applying the «ObjectStateMachine» is assigned to the *Book* object to represent its behaviour. The *Book* object appears in Figure 5.4, while its behavioural state machine diagram is represented in Figure 5.5. The Java source code for the *Book* object in listing 5.2 is produced by running the generator. The *Book* class shows the Java class file declaration and the attributes and operations in the class. It also shows the *handleEvent* operation, which represents the real implementation of the *Book* behaviour, which, in turn, corresponds to the implementation of the *BookStatus* state machine, as specified in the UP4WS (see Section 3.3.1.3.3).

```

1 package sample.webservice.data;
2
3 import java.util.*;

```

```
4 public class Book {
5
6     private String title;
7
8     private String author;
9
10    private String isbn;
11
12    private BookStatusStates currentState = BookStatusStates.AVAILABLE;
13
14    public void setCurrentState(BookStatusStates currentState) {
15        this.currentState = currentState;
16    }
17    public BookStatusStates getCurrentState() {
18
19        return this.currentState;
20    }
21
22    public void setTitle(String title) {
23        this.title = title;
24    }
25    public String getTitle() {
26
27        return this.title;
28    }
29
30    public void setAuthor(String author) {
31        this.author = author;
32    }
33    public String getAuthor() {
34
35        return this.author;
36    }
37
38    public void setIsbn(String isbn) {
39        this.isbn = isbn;
40    }
41    public String getIsbn() {
42
43        return this.isbn;
44    }
45
46
47    public String getBookInfo() {
48
49        return new String("Title : " + this.getTitle() + " isbn : "
50            + this.getIsbn() + " Author : " + this.getAuthor()
51            + "status : " + this.getCurrentState());
52    }
53
54    public void handleEvent(BookStatusEvents event) {
55        switch (currentState) {
56            case AVAILABLE :
57                if (event == BookStatusEvents.LENDBOOK) {
58
59                    currentState = BookStatusStates.ONLOAN;
60
61                    break;
62                }
63            }
64    }
65 }
```

```
63
64     throw new IllegalStateException("Cannot handle event " + event
65         + " for state " + currentState);
66
67     case NOTAVAILABLE :
68         if (event == BookStatusEvents.ADDBOOK) {
69
70             currentState = BookStatusStates.AVAILABLE;
71
72             break;
73         }
74
75         throw new IllegalStateException("Cannot handle event " + event
76             + " for state " + currentState);
77
78     case ONLOAN :
79         if (event == BookStatusEvents.RETURNBOOK) {
80
81             currentState = BookStatusStates.AVAILABLE;
82
83             break;
84         }
85
86         throw new IllegalStateException("Cannot handle event " + event
87             + " for state " + currentState);
88
89     }
90 }
91 }
```

Listing 5.2: Book Object Java Source Code

5.4.1.2 Generating the Configuration Files

The configuration files are platform dependent files needed to deploy and implement the library system Web service and any other Web service application and are required for the platforms that are based on the Apache Axis2/Java Web service engine. The files in this respect are the *build.xml* and the *services.xml* files. The *build.xml* file enables the compilation of the Java files at the service provider side the generation of the archive file *LibrarySystemService.aar*, which is uploaded to the Apache Tomcat application server in order to deploy the Web service and enable its execution. The *services.xml* file is required to enable the deployment of the Web service. Listing 5.3 shows the *services.xml* file for the library system Web service. The *services.xml* files appears in the screen shot as part of the generated output in Figure 5.7.

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2 <service name="LibrarySystemService" scope="application">
3
4     <description>
5         Library System Web Service
6     </description>
```

```

7      <messageReceivers>
8          <messageReceiver
9              class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"
10             mep="http://www.w3.org/2004/08/wsdl/in-only"/>
11
12             <messageReceiver
13                 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"
14                 mep="http://www.w3.org/2004/08/wsdl/in-out"/>
15         </messageReceivers>
16
17         <parameter
18             name="ServiceClass">sample.webservice.service.LibrarySystemService
19         </parameter>
20 </service>

```

Listing 5.3: *services.xml* File for the Library System Web Service

5.4.1.3 Generating the README.txt File

The *README.txt* file is a documentation file that aims at giving the user instructions on how to execute the Web service application. As specified in Section 3.3.2.5, the *README.txt* file mainly provides information about the Web service execution. Listing 5.4 represents the *README.txt* file that results from running the generator.

```

1
2 Introduction
3 =====
4 This case study (LibrarySystemService Web Service) has been implemented to validate
5 the proposed UML profile for Web services and the code generation process.
6 The case study shows a complete working example that represents the modelling
7 of Web services with UML, and the model transformation and code generation of
8 source code and configuration files for the relevant platform.
9 In this case study, both perspectives of service provider and service client has been introduced.
10
11 Web Services using Apache Axis2
12 =====
13 This program contains the source code for the LibrarySystemService Web service.
14 This source code is resulted from a UML model developed using a UML Tool,
15 and transformed into a java code using Xpand language together with some Xtend
16 functions in Eclipse.
17
18 Prerequisites and Environment Settings
19 =====
20 1. Windows as an Operating System (The case study has been implemented on
21 Windows XP and Windows Vista)
22 2. Apache Ant 1.6.2 or later
23 3. Apache Axis2
24 4. Apache Tomcat
25 5. Java
26
27 How to deploy the service on the Application Server?
28 =====
29 1. Compile the server side (service provider side) classes with ANT.
30 2. Put the LibrarySystemService.aar inside the services folder in Tomcat
31 (... \apache-tomcat\webapps\axis2\WEB-INF\services)
32 3. Initiate the tomcat server by typing the relevant command, e.g. tomcat.exe or startup.bat

```



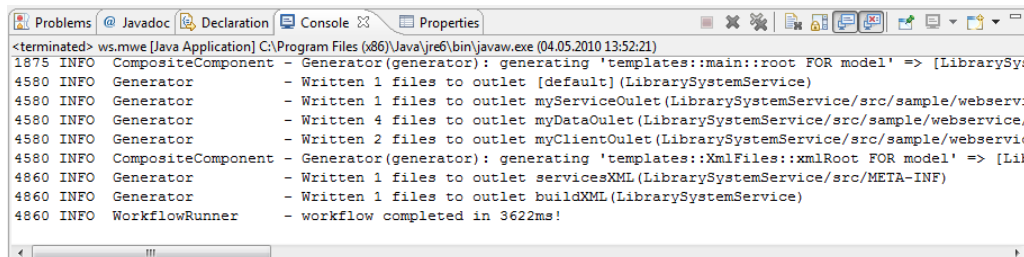
```

33 from inside the bin folder.
34 4. Make sure that the service is correctly deployed by typing
35 http://localhost:8080/axis2/services/LibrarySystemService?wsdl in your browser
36
37 Compile the Client side Classes by typing the following commands:
38 =====
39
40 First go to the root, where the build.xml file resides:
41
42 javac -sourcepath src\sample\webservice\client -classpath build\LibrarySystemService
43 -extdirs "C:\ApacheInstallation\axis2-1.5\lib" -d build\LibrarySystemService
44 src\sample\webservice\client\LibrarySystemServiceClient.java
45
46 javac -sourcepath src\sample\webservice\client -classpath build\LibrarySystemService
47 -extdirs "C:\ApacheInstallation\axis2-1.5\lib" -d build\LibrarySystemService
48 src\sample\webservice\rpcclient\LibrarySystemServiceClientMain.java
49
50 Running the Client Side classes by typing the following commands:
51 =====
52
53 First go to the root which ends with ...\build\LibrarySystemService
54 N.B. The name of the folder that contains your Web service files must hold the same name of
55 your Web service.
56
57 java -Djava.ext.dirs="C:\ApacheInstallation\axis2-1.5\lib"
58 sample.webservice.client.LibrarySystemServiceClientMain
59
60 Help
61 ===
62 Please contact (wdahman@informatik.uni-goettingen.de or arrivalsw@yahoo.com)
63 if you have any troubles running the Web service.

```

Listing 5.4: Library System Web Service *README.txt* File

The successful run of the generator is shown in the screen shots in Figures 5.6 and 5.7. The screen shot in Figure 5.7 shows errors in the output Java files. These errors do not reflect incorrectness in the source code, but they are resulted from the irrelevant position of the Java files. As soon as the files move to the execution environment, all errors will be resolved.



```

<terminated> ws.mwe [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (04.05.2010 13:52:21)
1875 INFO CompositeComponent - Generator(generator): generating 'templates::main::root FOR model' => [LibrarySy:
4580 INFO Generator - Written 1 files to outlet [default] (LibrarySystemService)
4580 INFO Generator - Written 1 files to outlet myServiceOutlet (LibrarySystemService/src/sample/webserv:
4580 INFO Generator - Written 4 files to outlet myDataOutlet (LibrarySystemService/src/sample/webservice,
4580 INFO Generator - Written 2 files to outlet myClientOutlet (LibrarySystemService/src/sample/webservi:
4580 INFO CompositeComponent - Generator(generator): generating 'templates::XmlFiles::xmlRoot FOR model' => [Lii
4860 INFO Generator - Written 1 files to outlet servicesXML (LibrarySystemService/src/META-INF)
4860 INFO Generator - Written 1 files to outlet buildXML (LibrarySystemService)
4860 INFO WorkflowRunner - workflow completed in 3622ms!

```

Figure 5.6: Code Generator Successful Execution

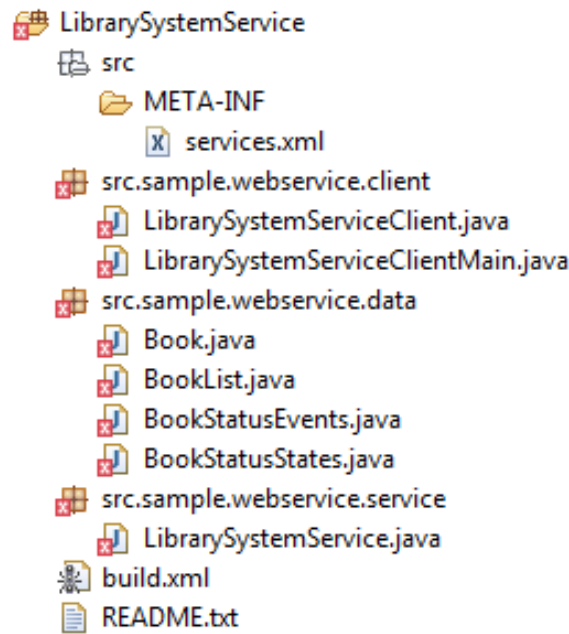


Figure 5.7: LibrarySystemService Folder After Generator Execution

5.4.2 Executing the Library System Web Service

Executing the Web service is concerned with running the library system Web service on the Apache Axis2/Java Web service engine. The execution is done by following the instructions stated in the *README.txt* file, which includes setting up the execution environment. At first, the Java and configuration files have to be copied in the execution environment. The Java files in the service provider side are compiled first in order to generate the service archive file (*LibrarySystemService.aar*). This file is responsible for deploying the library system Web service on the Apache Tomcat application server. The Java clients at the service client side are then compiled. The *LibrarySystemServiceClientMain.java* can be executed to run the library system Web service. Appendix F presents detailed information about executing the Web service.

5.5 Summary

This chapter introduced the realisation of the UP4WS and WSDM from Chapters 3, and 4. It showed a real implementation of Web services via the library system Web service case study. The case study has proven the powerful UML customisation by means of UML profiles and the possibility to generate source code and different types of textual files from UML models. However, in some cases it was not possible to generate some constructs of the target programming language

(e.g. Java HashMap). The library system Web service has been modelled using the Magic Draw UML tool, which can export UML models in the correct format for the Eclipse/Xpand generator. Other UML tools need additional adaptation to produce the format, that is understandable by the used Eclipse/Xpand generator. The Apache Axis2/Java together with the Apache Tomcat have been used for the implementation and deployment of the Web service. The examples provided by the Apache Axis2 distribution give different types of Web services implementations. The implementation of library system Web services represents a general example that should enable the implementation of any Web service on the Apache Axis2/Java engine.

Chapter 6

Conclusion

This chapter summarises the work performed in this thesis. It also presents an outlook illustrating the future perspectives of this work, and its possible extensions.

6.1 Summary

The thesis defines an approach for the development of executable Web services. The proposed approach is composed of a sequence of steps to produce an executable Web service source code. The work includes a specification of a platform for the implementation and deployment of Web services. The executable code is generated from UML models which form the starting point in the code generation process.

At first, a UML Profile for Web Services (**UP4WS**) is defined. The **UP4WS** defines two types of extensions for Web services. The first type is the Web service specific extensions which are mandatory and must be specified for any Web service application disregarding on which platform the Web service will be implemented and deployed. Those extensions consider the service provider and service client sides, since the Web service can play the role of the service provider and service client at the same time. The second type is the extensions that enable the generation of executable source code for Web services. Those extensions enable basically the implementation of Web services behaviour by generating the source code from **UML** state machine diagrams. In addition, they serve to generate or complete the generation of some parts of the source code, such as attribute declarations. The thesis defines the **UP4WS** implementation guidelines together with the transformation rules for code generation. The transformation rules are defined in the Xpand language. The second step of the proposed approach is

the definition of the Web Services Development Model (**WSDM**). The **WSDM** defines a set of tasks that shall be followed to develop the model for the selected Web service. The **WSDM** consists of three tasks in its life cycle:

- **The Web Services Requirements Analysis:** in the requirements analysis task, the requirements for the Web service shall be analysed in order to identify the general goal for the Web service and the exact service that the Web service shall provide. At this task, the use case and class diagrams to model the requirements and the initial architecture for the Web service.
- **The Web Services Design:** in the design task, the initial architecture for the Web service is refined and extended to include the complete specification for the Web service. The allocation of the basic **UML** extensions defined in the **UP4WS** must take place in this task as well. Class diagrams are used to represent the Web service architecture, while state machine diagrams are used for modelling the behaviour of the Web service.
- **The Web Services Implementation:** the implementation task is concerned with explaining how to transform the model and how to deploy and implement the Web service. This contains specifying the plan for implementing the Web service. The plan includes different specifications considering the generator, the platform for executing the Web service and its settings.

For the validation of the entire work, the thesis presents a case study to implement a library system Web service. Running and executing the Web service requires setting and installation of the platform, which is the Apache Axis2/Java as a Web service engine, and the Apache Tomcat as the application server to host the Web service. A full and complete description on how to set the platform environment is presented in the thesis, and also in the README.txt file which is generated together with the Java source code and the platform dependent files.

6.2 Discussion

The work described in this thesis is an important step towards the generation of executable source code from **UML** models and profiles. The thesis has proven the flexibility of **UML** in modelling specific domain requirements by means of its execution mechanism. It has also shown the efficiency of **UML** models in representing the dynamic behaviour of software applications. The flexibility of **UML** models, and its extension mechanism have enabled an effective definition of the transformation rules for the code generation process. In comparison with the related work, this thesis has introduced a comprehensive approach for the development of executable Web services, and not only for **WSDL** documents.

Furthermore, the thesis has presented a complete set of transformation rules in order to generate the output files from the **UML** model and profile. The transformation rules run on the Xpand/Eclipse generator which, despite its maturity, has proven efficiency in generating consistent output.

6.3 Outlook

The work in this thesis enables various future work. Below is a description of worthwhile future research directions.

This thesis showed how to use state machine diagrams for the representation of Web services behaviour. Using other behavioural diagrams, such as activity diagrams is promising as well and could be investigated in future work. Furthermore, this thesis demonstrated how to generate Java source code for a specific platform and Web service engine. It should be easily possible to extend this approach to the implementation of Web services in other programming languages and on different platforms and Web service engines.

The composition of Web services is one of the most important topics related to Web services. Composite Web services enable the reuse of already existing Web services to provide more comprehensive and larger Web services. In such cases, the Web service plays the role of service provider and service client at the same time. Therefore, it should have the mechanisms for providing and invoking other Web services. The **UP4WS** defines a specific set of **UML** extensions for a single Web service. Since the composite Web services have more sophisticated architecture and behaviour, it may be necessary to define additional **UML** extensions for composite Web services. The thesis provides a promising starting point for investigating the required modifications of the profile definition and its implementation.

Another important extension of this work is the field of Grid Web services, which is referred to as Grid services. The community of Grids is now adopting or developing several standards in connection with Web services such as Open Grid Services Infrastructure (**OGSI**) [**OGS03**], and WS-Resource Framework (**WSRF**) [**WSR04**]. Further investigations are needed to identify the novelty of Grid Web services and how they can be developed by means of model transformation techniques. Just by adding the definition of relevant **UML** extensions and their implementation using the appropriate transformation rules, the results of this thesis should be applicable as well to support the domain of emerging Grid services.

Bibliography

- [AA2a] Apache Axis2/C. <http://ws.apache.org/axis2/c/>, Last visited: June 2010. [cited at p. 143]
- [AA2b] Apache Axis2 Installation Guide. http://ws.apache.org/axis2/1_5_1/installationguide.html, Last visited: June 2010. [cited at p. 144]
- [AA2c] Apache Axis2/Java. <http://ws.apache.org/axis2/>, Last visited: June 2010. [cited at p. 143]
- [AN05] J. Arlow and I. Neustadt. *UML 2 and the Unified Process*. Addison Wesley, second edition, June 2005. [cited at p. 12, 13, 14, 17]
- [ANT] The Apache ANT Project. <http://ant.apache.org/>, Last visited: June 2010. [cited at p. 63]
- [Arm02] C. Armstrong. Modeling Web Services with UML. OMG Web Services Workshop 2002, 2002. [cited at p. 4]
- [ATO] Apache Tomcat 6.0 Setup. <http://tomcat.apache.org/tomcat-6.0-doc/setup.html>, Last visited: June 2010. [cited at p. 144]
- [BOM] The Booch Methodology. <http://infolab.stanford.edu/~burback/watersluice/node55.html>, Last visited: June 2010. [cited at p. 12]
- [BPE03] BPEL4WS V1.1 specification. <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf>, Last visited: June 2010, 5 May 2003. [cited at p. 5]
- [CH06] K. Czarnecki and S. Helsen. Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–645, 2006. [cited at p. 21, 22]
- [Con03] J. Conallen. *Building Web Application with UML*. Addison Wesley, second edition, 2003. [cited at p. 9, 11]
- [COR02] UML Profile for CORBA, v 1.0. formal-02-04-01, April 2002. [cited at p. 9, 10, 16]
- [CWM03] Common Warehouse Metamodel, v1.1. formal/2003-03-02, March 2003. [cited at p. 17]

- [DDDT03] H. M. Deitel, P. J. Deitel, B. DuWaldt, and L. K. Trees. *Web Services A Technical Introduction*. Prentice Hall, 2003. [cited at p. 10, 11]
- [Dol04] K. Dollard. *Code Generation in Microsoft .NET*. Apress, 2004. [cited at p. 22]
- [Erl04] T. Erl. *Service-Oriented Architecture. A Field Guide to Integrating XML and Web Services*. Prentice Hall, 2004. [cited at p. 9]
- [Fow04] M. Fowler. *UML Distilled*. Addison Wesley, 2004. [cited at p. 12, 18]
- [Fra03] D. S. Frankel. *Model Driven Architecture, Applying MDA to Enterprise Computing*. Wiley Publishing, Inc., 2003. [cited at p. 21]
- [FV04] L. Fuentes and A. Vallecillo. An Introduction to UML Profiles. *UP-GRADE the European Journal for the Informatics Professional*, V(2), 2004. [cited at p. 16, 19]
- [GSSO04] R. Gronme, D. Skogan, I. Solheim, and J. Oldevik. Model-Driven Web Services Development. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (IEEE'04)*, pages 42–45, Washington, DC, USA, 2004. IEEE Computer Society. [cited at p. 4, 5]
- [Her03] J. Herrington. *Code Generation in Action*. Manning, 2003. [cited at p. 22, 24]
- [JBR03] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison Wesley, August 2003. [cited at p. 2, 13, 15]
- [JBR04] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Modeling Language Reference Manual*. Addison Wesley, second edition, 2004. [cited at p. 13, 14, 15, 18]
- [KT08] S. Kelly and J. P. Tolvanen. *Domain Specific Modeling*. Wiley-Interscience Publication, 2008. [cited at p. 22, 23]
- [KWB03] A. Kleppe, J. Warmer, and W. Best. *MDA Explained, The Model-Driven Architecture: Practice and Promise*. Addison Wesley, 2003. [cited at p. 21]
- [MD] Magicdraw. <http://www.nomagic.com>, Last visited: June 2010. [cited at p. 81]
- [MDA03] MDA Guide, v1.0.1. [omg/2003-06-01](http://www.omg.org/2003-06-01), June 2003. [cited at p. 2, 19, 20, 21]
- [MG06] T. Mens and P. Van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006. [cited at p. 22]
- [Mil03] J. Miller. MDA Model Driven Architecture. Fourth Workshop On UML for Enterprise Applications: Delivering the Promise of MDA, June 23-26, 2003 - Burlingame, California, USA, 2003. [cited at p. 21]
- [New02] E. Newcomer. *Understanding Web Services*. Addison Wesley, 2002. [cited at p. 11]
- [OAW] OpenArchitectureWare (oAW). <http://www.openarchitectureware.org>, Last visited: June 2010. [cited at p. 25]

- [OAW10] OpenArchitectureWare User Guide. Version 4.3.1, 15 December, 2010. [cited at p. 23, 25]
- [OCL06] Object Constraint Language Specification, v2.0. formal/2006-05-01, May 2006. [cited at p. 18]
- [OGS03] Open Grid Services Infrastructure (OGSI), Version 1.0 Status. GFD-R-P.15, Last visited: June 2010, June 27, 2003. [cited at p. 101]
- [OMG] Object Management Group (OMG). <http://www.omg.org/>, Last visited: June 2010. [cited at p. 2, 12]
- [OMT] Object Modeling Technique. <http://infolab.stanford.edu/~burback/watersluice/node56.html>, Last visited: June 2010. [cited at p. 12]
- [ROM] Rational Objectory Methodology. <http://infolab.stanford.edu/~burback/watersluice/node57.html>, Last visited: June 2010. [cited at p. 12]
- [Sam02] M. Samek. *Practical Statecharts in C/C++*. CMP Books, 2002. [cited at p. 42]
- [Sco04] K. Scott. *Fast Track UML 2.0*. Apress, 2004. [cited at p. 14]
- [SCV03] S.Marcos, V.de Castro, and B. Vela. Representing Web Services with UML: A Case Study. In *UNITN - The First International Conference on Service Oriented Computing*. UNITN, December 2003. [cited at p. 4]
- [SGS04] D. Skogan, R. Gronmo, and I. Solheim. Web Service Composition in UML. In *Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04)*, pages 47–57, Washington, DC, USA, 2004. IEEE Computer Society. [cited at p. 4, 5]
- [SOA07] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). <http://www.w3.org/TR/soap12-part1/>, Last visited: June 2010, W3C Recommendation 27 April 2007. [cited at p. 11]
- [SV06] T. Stahl and M. Voelter. *Model-Driven Software Development*. John Wiley & Sons, Ltd, 2006. [cited at p. 22]
- [TDE03] S. Thöne, R. Depke, and G. Engels. Process-Oriented, Flexible Composition of Web Services with UML. In *Advanced Conceptual Modeling Techniques*, pages 390–401. SpringerLink, October 2003. [cited at p. 4, 5]
- [Til07] S. Tilkov. Overview: Web Services Standards and Specifications. <http://www.innoq.com/resources/WebServicesStandardsOverview-2005-01.pdf>, Last visited: June 2010, February 2007. [cited at p. 11]
- [UDD04] UDDI Version V3.0.2. <http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm>, Last visited: June 2010, UDDI Spec Technical Committee Draft, Dated 19 October 2004. [cited at p. 11]
- [UMLa] Unified Modeling Language. <http://www.uml.org/>, Last visited: June 2010. [cited at p. 1]

- [UMLb] Eclipse UMLX Project. <http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/UMLX/index.html>, Last visited: June 2010. [cited at p. 12]
- [UML10a] Unified Modeling Language: Infrastructure, V2.3. formal/2010-05-03, May 2010. [cited at p. 1]
- [UML10b] Unified Modeling Language: Superstructure, V2.3. formal/2010-05-05, May 2010. [cited at p. 1, 13, 18, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 47, 48]
- [UN09] M. Usman and A. Nadeem. Automatic Generation of Java Code from UML Diagrams using UJECTOR. *International Journal of Software Engineering and Its Applications*, 3(2), 2009. [cited at p. 4, 5]
- [VKEH06] M. Voelter, B. Kolb, S. Efftingo, and A. Haase. From Front End To Code - MDS in Practice. <http://www.eclipse.org/articles/Article-FromFrontendToCode-MSDInPractice/article.html>, Last visited: June 2010, June 15 2006. [cited at p. 42]
- [VMX06] Documentation of V-Modell XT, 2006. [cited at p. 2]
- [W3C07] World Wide Web Consortium (W3C). <http://www.w3.org>, Last visited: June 2010, 2007. [cited at p. 9]
- [WSA] Web Services Addressing (WS-Addressing) 10 August 2004. <http://www.w3.org/Submission/ws-addressing/>, Last visited: June 2010. [cited at p. 144]
- [WSD07] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. <http://www.w3.org/TR/wsdl20/>, Last visited: June 2010, W3C Recommendation 26 June 2007. [cited at p. 11]
- [WSR] OASIS Web Services Reliable Messaging (WSRM) TC 15 November 2004. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrm, Last visited: June 2010. [cited at p. 144]
- [WSR04] The WS-Resource Framework (WSRF), Version 1.2. <http://www.globus.org/wsrp/>, Last visited: June 2010, January 20, 2004. [cited at p. 101]
- [WSS07] Web Services Standards Overview. <http://www.innoq.com/soa/ws-standards/poster/innoQWS-StandardsPoster2007-02.pdf>, Last visited: June 2010, February 2007. [cited at p. 11]
- [XML] Extensible Markup Language (XML). <http://www.w3.org/XML/>, Last visited: June 2010. [cited at p. 1]
- [XPA] Eclipse.org Xpand. , Last visited: June 2010. [cited at p. 3, 26, 63]

Appendices

Appendix A

Case Study Model

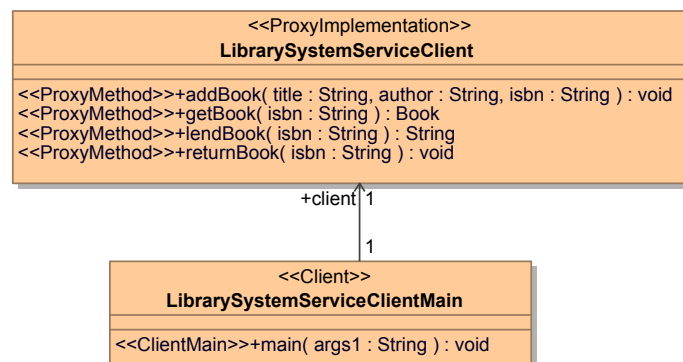


Figure A.1: Class Diagram at Service Client Side

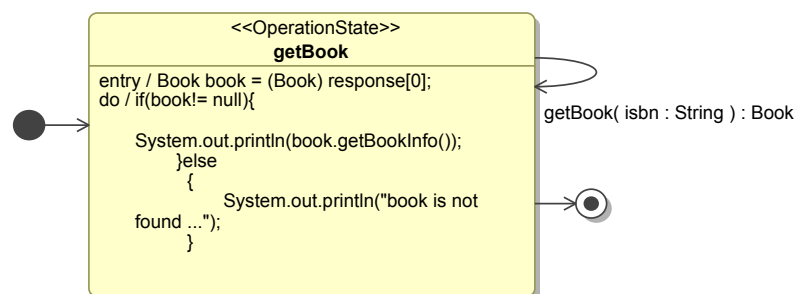


Figure A.2: `getBookInLibrarySystemServiceClient`

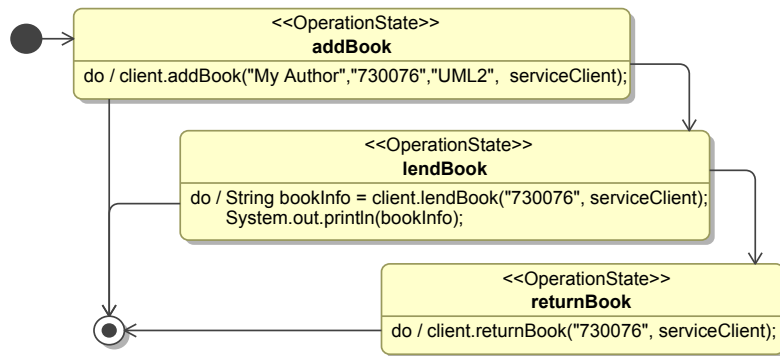


Figure A.3: mainInLibrarySystemServiceClientMain

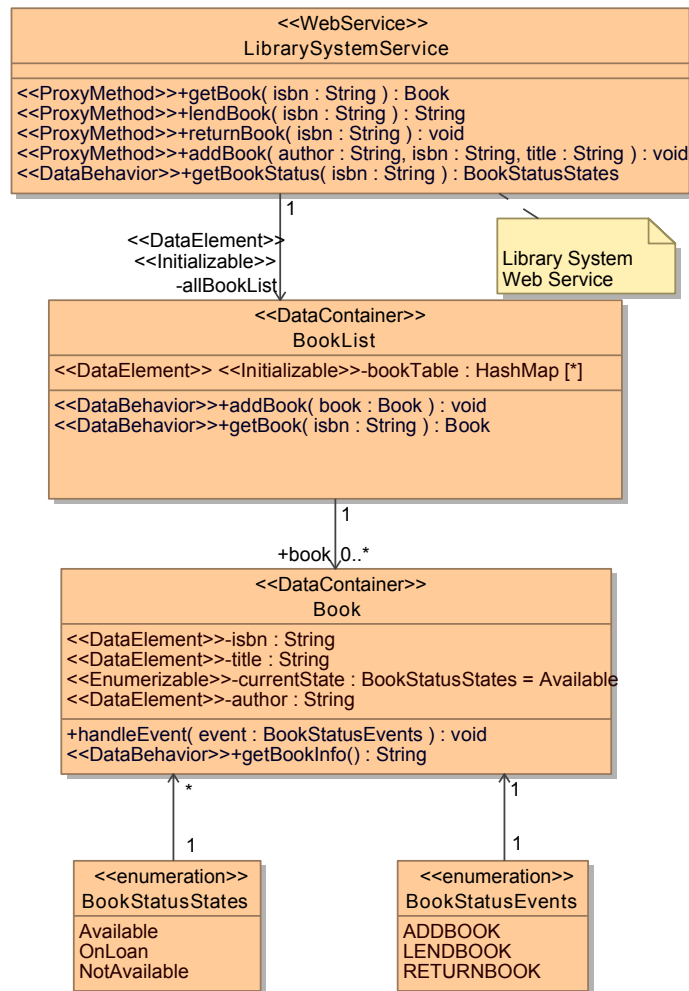


Figure A.4: Class Diagram at Service Provider Side

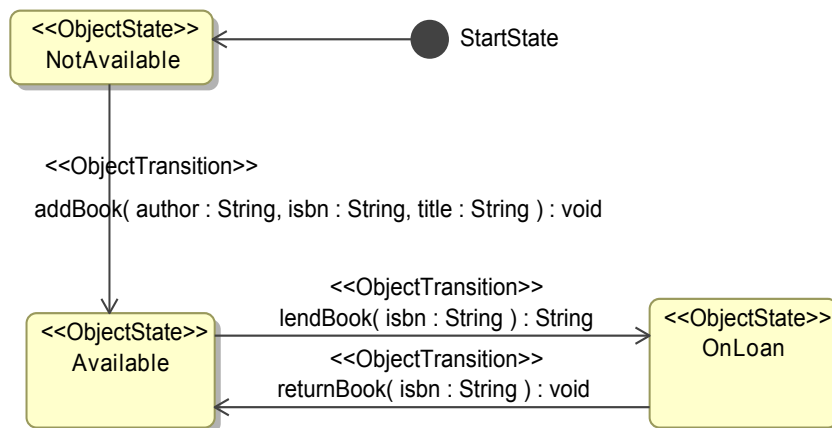


Figure A.5: BookStatus

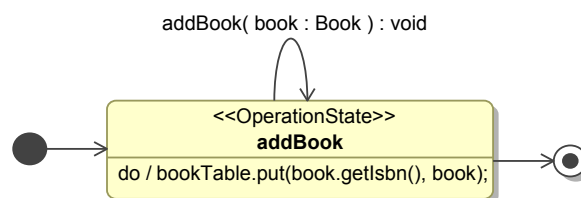


Figure A.6: addBookInBookList

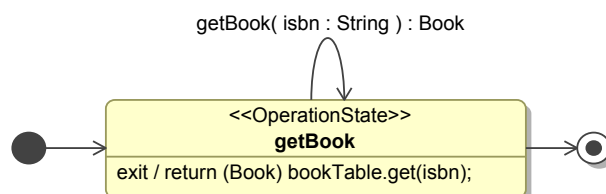


Figure A.7: getBookInBookList

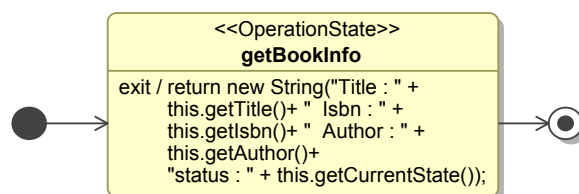


Figure A.8: getBookInfoInBook

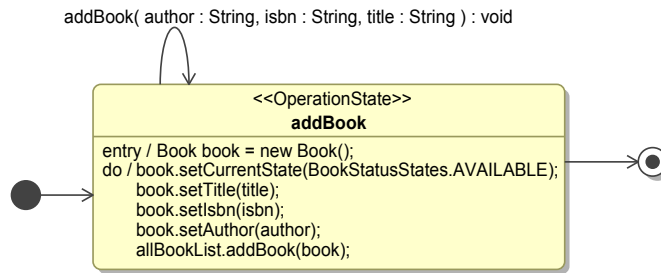


Figure A.9: addBookInLibrarySystemService

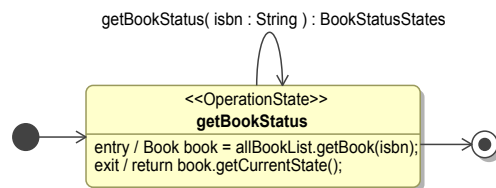


Figure A.10: getBookStatusInLibrarySystemService

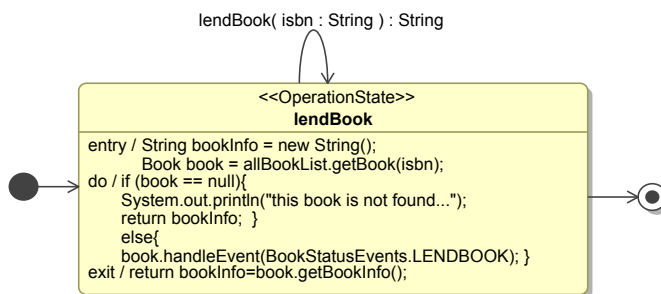


Figure A.11: lendBookInLibrarySystemService

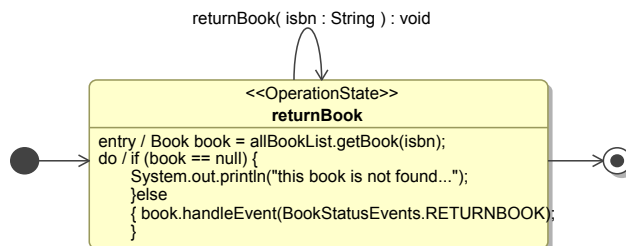


Figure A.12: returnBookInLibrarySystemService

Appendix B

Transformation Rules for Generating Java Code

B.1 CommonTemplate Template File

```
1 «IMPORT UP4WS»
2
3 «EXTENSION templates::MyExtensions»
4
5 «REM»-----«ENDREM»
6
7 «DEFINE createImport FOR uml::Type»
8   import «packageName()-».«"*"-»;
9 «ENDDEFINE»
10
11 «REM»-----«ENDREM»
12
13 «DEFINE createImport FOR uml::PrimitiveType»
14 «IF name.hasJavaPath()-»
15   import «name.javaPath()-».«"*"-»;
16 «ENDIF»
17 «ENDDEFINE»
18
19 «REM»-----«ENDREM»
20
21 «DEFINE createDataElement FOR UP4WS::DataElement»
22
23   «IF defaultValue == null && metaType.name
24     != "UP4WS::DataElement,UP4WS::Initializable"»
25
26     «visibility» «type.name» «name»;
27   «ELSE»
28     «IF defaultValue == null && metaType.name
29       == "UP4WS::DataElement,UP4WS::Initializable"»
30
31     «visibility» «type.name» «name» = new «type.name»();
32   «ENDIF»
```

```

33
34     «IF defaultValue != null && type.name == "String"»
35
36         «visibility» «type.name» «name» = "«value()»";
37
38     «ELSE»
39
40         «IF defaultValue != null && type.name != "String"»
41
42             «visibility» «type.name» «name» = «value()»;
43
44         «ENDIF»
45     «ENDIF»
46 «ENDDEFINE»
47 «ENDDEFINE»
48
49 «REM»-----«ENDREM»
50
51 «DEFINE initializable FOR UP4WS::Initializable»
52     «name-» = new «type.name-»();
53 «ENDDEFINE»
54
55 «REM»-----«ENDREM»
56
57 «DEFINE enumerizable FOR UP4WS::Enumerizable»
58     «IF defaultValue == null-»
59         «visibility» «type.name» «name-»;
60     «ELSE»
61         «visibility» «type.name» «name» =
62             «type.name».«value().toUpperCase()-»;
63     «ENDIF-»
64 «ENDDEFINE»
65
66 «REM»-----«ENDREM»
67
68 «DEFINE gettersAndSetters FOR UP4WS::DataElement»
69     «IF visibility.toString() == "private"»
70     public void set«name.toFirstUpper()»
71         («type.name» «name.toFirstLower()»){
72
73         this.«name» = «name.toFirstLower()»;
74     }
75     public «type.name» get«name.toFirstUpper()» (){
76         return this.«name»;
77     }
78     «ENDIF»
79 «ENDDEFINE»
80
81 «REM»-----«ENDREM»
82
83 «DEFINE gettersAndSetters FOR UP4WS::Enumerizable»
84     «IF visibility.toString() == "private"»
85     public void set«name.toFirstUpper()»
86         («type.name» «name.toFirstLower()»){
87
88         this.«name» = «name.toFirstLower()»;
89     }
90     public «type.name» get«name.toFirstUpper()» (){
91         return this.«name»;

```

```

92     }
93     «ENDIF»
94 «ENDDEFINE»
95
96 «REM»-----«ENDREM»
97 «DEFINE methodTpl FOR UP4WS::DataBehavior»
98     «visibility» «IF isStatic» static «ENDIF» «IF type==null» void
99     «ELSE» «type.name» «ENDIF» «name» («FOREACH getParameters()
100 AS p SEPARATOR ", » «p.type.name.toJava()» «p.name» «ENDFOREACH»){
101     «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
102     (UP4WS::OperationStateMachine) AS wssm-»
103     «IF name+"In"+owner.name == wssm.name-»
104     «EXPAND implementState(wssm) FOREACH
105     wssm.eAllContents.typeSelect(UP4WS::OperationState)-»
106     «ENDIF-»
107     «ENDFOREACH-»
108     }
109 «ENDDEFINE»
110 «REM»-----«ENDREM»
111
112 «DEFINE implementState (UP4WS::OperationStateMachine sm)
113     FOR UP4WS::OperationState»
114
115     «IF entry != null-»
116     «entry.name-»
117     «ENDIF-»
118
119     «IF doActivity != null-»
120     «doActivity.name-»
121     «ENDIF»
122
123     «IF exit != null-»
124     «exit.name-»
125     «ENDIF-»
126
127 «ENDDEFINE»
128
129 «REM»-----«ENDREM»
130
131 «DEFINE interfaceTpl FOR UP4WS::Interface»
132     «FILE name + ".java" myDataOutlet-»
133     package «packageName(this)»;
134
135     «EXPAND createImport FOREACH usedInterfaceTypes()»
136
137     public interface «name» {
138     «EXPAND interfaceAttTpl
139     FOREACH eAllContents.typeSelect(UP4WS::InterfaceElement)»
140     «EXPAND interfaceOpTpl
141     FOREACH eAllContents.typeSelect(UP4WS::InterfaceMethod)»
142     }
143     «ENDFILE»
144 «ENDDEFINE»
145
146 «REM»-----«ENDREM»
147
148 «DEFINE interfaceAttTpl FOR UP4WS::InterfaceElement»
149     «IF type.name == "String"»
150     final «IF isStatic-» static «ENDIF-» «type.name-» «name-» = "«value()-»";

```

```

151     «ELSE-»
152     final «IF isStatic-» static «ENDIF-» «type.name-» «name-» = «value()-»;
153     «ENDIF-»
154 «ENDDEFINE»
155
156 «REM»-----«ENDREM»
157
158 «DEFINE interfaceOpTpl FOR UP4WS::InterfaceMethod»
159     abstract «IF type==null» void
160         «ELSE» «type.name-» «ENDIF-» «name-»
161         («FOREACH getParameters() AS p SEPARATOR ", "-»
162         «p.type.name.toJava()-» «p.name-»«ENDFOREACH-»);
163 «ENDDEFINE»
164
165 «REM»-----«ENDREM»
166
167 «DEFINE Events FOR UP4WS::ObjectStateMachine»
168     «IF owner.metaType.name == "UP4WS::DataContainer"»
169     «FILE eventsEnumFileName() myDataOutlet-»
170
171     package «packageName(this)»;
172     public enum «eventsEnumName()» {
173
174         «FOREACH events().constantName().toSet() AS s SEPARATOR ", "-»
175         «s-»
176         «ENDFOREACH»
177     }
178     «ENDFILE»
179     «ENDIF»
180 «ENDDEFINE»
181
182 «REM»-----«ENDREM»
183
184 «DEFINE States FOR UP4WS::ObjectStateMachine»
185     «IF owner.metaType.name == "UP4WS::DataContainer"»
186     «FILE statesEnumFileName()myDataOutlet-»
187     package «packageName(this)»;
188     public enum «statesEnumName()»{
189         «FOREACH states() AS s SEPARATOR ", "-»
190         «s.constantName()»
191         «ENDFOREACH»
192     }
193     «ENDFILE»
194     «ENDIF»
195 «ENDDEFINE»

```

B.2 DataContainer Template File

```

1 «IMPORT uml»
2
3 «EXTENSION templates::MyExtensions»
4
5 «REM»-----«ENDREM»
6
7 «DEFINE dataContainerRoot FOR UP4WS::DataContainer»
8     «EXPAND createDataContainer»
9 «ENDDEFINE»
10

```

```

11 «REM»-----«ENDREM»
12
13 «DEFINE createDataContainer FOR UP4WS::DataContainer»
14
15     «FILE name + ".java" myDataOutlet-»
16
17     package «packageName(this)»;
18
19     «EXPAND CommonTemplates::createImport FOREACH usedTypes()»
20
21     «visibility-» «IF isAbstract-» abstract «ENDIF-» class «name»
22     «IF !superClass.isEmpty-» extends «superClass.first().name-»«ENDIF-»
23     «IF this.interfaceRealization.contract.size > 0-» implements
24     «FOREACH this.interfaceRealization.contract AS i SEPARATOR ","-»
25     «i.name-»«ENDFOREACH-»«ENDIF-»{
26
27         «EXPAND CommonTemplates::createDataElement
28         FOREACH eAllContents.typeSelect(UP4WS::DataElement)-»
29         «EXPAND CommonTemplates::enumerizable
30         FOREACH eAllContents.typeSelect(UP4WS::Enumerizable)-»
31         «EXPAND CommonTemplates::gettersAndSetters
32         FOREACH eAllContents.typeSelect(UP4WS::Enumerizable)-»
33         «EXPAND CommonTemplates::gettersAndSetters
34         FOREACH eAllContents.typeSelect(UP4WS::DataElement)-»
35         «EXPAND CommonTemplates::methodTpl
36         FOREACH eAllContents.typeSelect(UP4WS::DataBehavior)-»
37         «EXPAND classBehavior
38         FOREACH eAllContents.typeSelect(UP4WS::ObjectStateMachine)-»
39         }
40     «ENDFILE»
41 «ENDDEFINE»
42
43 «REM»-----«ENDREM»
44
45 «DEFINE classBehavior FOR UP4WS::ObjectStateMachine»
46     «IF name == getOwnerName(this)+"Status" &&
47     owner.metaType.name == "UP4WS::DataContainer"»
48
49     public void handleEvent(«eventsEnumName(this)» event){
50     switch (currentState) {
51         «FOREACH this.states().reject
52         (s|UP4WS::ObjectFinalState.isInstance(s)) AS s-»
53
54         case «s.constantName()»:
55         «FOREACH s.outTransitionsWithEventTrigger() AS t-»
56
57             «IF t.trigger.event !=null-»
58             «FOREACH t.trigger.event AS e-»
59             if (event == «e.eventId(this)») {
60             «EXPAND executeTransitionForHandleEvent(this) FOR t»
61             break;
62             }«ENDFOREACH»«ENDIF»«ENDFOREACH»
63             «EXPAND illegalTransitionHandler»
64         «ENDFOREACH»
65     }
66     }
67     «ENDIF»
68 «ENDDEFINE»
69

```

```

70 «REM»-----«ENDREM»
71
72 «DEFINE executeTransitionForHandleEvent(UP4WS::ObjectStateMachine sm)
73   FOR UP4WS::ObjectTransition»
74
75   «IF UP4WS::ObjectState.isInstance(source) &&
76     ((UP4WS::ObjectState)source).exit!=null-»
77
78     «((UP4WS::ObjectState)source).exit.methodName()»;
79   «ENDIF-»
80   «FOREACH trigger.event AS e-»
81   «IF effect!=null-»
82   «effect.methodName()»;
83   «ENDIF»
84   «ENDFOREACH»
85   currentState = «target.stateId(sm)»;
86   «IF UP4WS::ObjectFinalState.isInstance(target)-»
87   terminated = true;
88   «ENDIF-»
89   «IF UP4WS::ObjectState.isInstance(target) &&
90     ((UP4WS::ObjectState)target).entry!=null-»
91   «((UP4WS::ObjectState)target).entry.methodName()»;
92   «ENDIF-»
93 «ENDDEFINE»
94
95 «REM»-----«ENDREM»
96
97 «DEFINE illegalTransitionHandler FOR UP4WS::ObjectStateMachine»
98   throw new IllegalStateException
99     ("Cannot handle event "+event+" for state "+currentState);
100 «ENDDEFINE»

```

B.3 WebService Template File

```

1 «IMPORT UP4WS»
2
3 «EXTENSION templates::MyExtensions»
4
5 «REM»-----«ENDREM»
6
7 «DEFINE serviceRoot FOR UP4WS::WebService»
8   «EXPAND createWebService»
9 «ENDDEFINE»
10
11 «REM»-----«ENDREM»
12
13 «DEFINE createWebService FOR UP4WS::WebService»
14   «FILE name + ".java" myServiceOutlet»
15   package «packageName(this)»;
16
17   «EXPAND CommonTemplates::createImport FOREACH usedTypes()»
18
19   «visibility-» class «name-» {
20     «EXPAND CommonTemplates::createDataElement
21     FOREACH eAllContents.typeSelect(UP4WS::DataElement)-»
22     «EXPAND CommonTemplates::enumerizable
23     FOREACH eAllContents.typeSelect(UP4WS::Enumerizable)-»
24     «EXPAND CommonTemplates::methodTmpl

```



```

25     FOREACH eAllContents.typeSelect(UP4WS::DataBehavior)-»
26     «EXPAND CommonTemplates::gettersAndSetters
27     FOREACH eAllContents.typeSelect(UP4WS::Enumerizable)-»
28     «EXPAND CommonTemplates::gettersAndSetters
29     FOREACH eAllContents.typeSelect(UP4WS::DataElement)-»
30     «EXPAND proxyMethodServiceTmpl
31     FOREACH eAllContents.typeSelect(UP4WS::ProxyMethod)-»
32     }
33     «ENDFILE»
34 «ENDDEFINE»
35
36 «REM»-----«ENDREM»
37
38 «DEFINE proxyMethodServiceTmpl FOR UP4WS::ProxyMethod»
39     «visibility-» «IF isStatic-» static «ENDIF-» «IF type==null-» void «ELSE-»
40     «type.name-» «ENDIF-» «name-» («FOREACH getParameters().sortBy(e.e.name)
41     AS p SEPARATOR ", "-» «p.type.name.toJava()-» «p.name-» «ENDFOREACH-»){
42
43     «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
44     (UP4WS::OperationStateMachine) AS wssm-»
45     «IF name+"In"+owner.name == wssm.name-»
46     «EXPAND CommonTemplates::implementState(wssm)
47     FOREACH wssm.checkStates()-»
48     «ENDIF-»
49     «ENDFOREACH-»
50     }
51 «ENDDEFINE»

```

B.4 Proxy Template File

```

1 «IMPORT UP4WS»
2
3 «EXTENSION templates::MyExtensions»
4 «EXTENSION org::eclipse::xtend::util::stdlib::counter»
5
6 «REM»-----«ENDREM»
7
8 «DEFINE proxyRoot FOR UP4WS::ProxyImplementation»
9     «EXPAND createProxyImpl-»
10 «ENDDEFINE»
11
12 «REM»-----«ENDREM»
13
14 «DEFINE createProxyImpl FOR UP4WS::ProxyImplementation»
15
16     «FILE name.toFirstUpper()+"_java" myClientOutlet-»
17
18     package «packageName(this)-»;
19
20     «EXPAND CommonTemplates::createImport FOREACH usedTypes()-»
21
22     import javax.xml.namespace.QName;
23     import org.apache.axis2.AxisFault;
24     import org.apache.axis2.addressing.EndpointReference;
25     import org.apache.axis2.client.Options;
26     import org.apache.axis2.rpc.client.RPCServiceClient;
27
28     «visibility» class «name.toFirstUpper()-» {

```

```

29     «EXPAND ProxyMethodClientTpl
30         FOREACH eAllContents.typeSelect(UP4WS::ProxyMethod)-»
31     }
32     «ENDFILE»
33 «ENDDEFINE»
34
35 «REM»-----«ENDREM»
36
37 «DEFINE ProxyMethodClientTpl FOR UP4WS::ProxyMethod»
38 «IF getVisibility(this) == "public"-»
39     «visibility-» «IF isStatic-» static «ENDIF-»
40     «IF type==null-» void «ELSE-» «type.name-» «ENDIF-» «name-»
41     («FOREACH getParameters().sortBy(e|e.name) AS p SEPARATOR ", "-»
42     «p.type.name.toJava()» «p.name-» «ENDFOREACH-»,
43     RPCServiceClient rpcClient) throws AxisFault{
44
45     QName opGet = new QName("http://service.webservice.sample", "«name-»");
46     rpcClient.getOptions().setAction("urn:«name-»");
47
48     Object[] args = new Object[«count()-1»];
49
50     «FOREACH getParameters().sortBy(e|e.name) AS p ITERATOR i-»
51     args[«i.counter0»]= «p.name-»;
52     «ENDFOREACH»
53
54     «IF type.name == "void"-»
55     rpcClient.invokeRobust(opGet, args);
56     «ELSE-»
57     Class[] returnTypes = new Class[] { «type.name-».class };
58     Object[] response = rpcClient.invokeBlocking(opGet, args, returnTypes);
59     «ENDIF-»
60
61     System.out.println("after «name-» invoke");
62
63     «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
64     (UP4WS::OperationStateMachine) AS wssm-»
65     «IF name+"In"+owner.name == wssm.name-»
66     «EXPAND CommonTemplates::implementState(wssm)
67     FOREACH wssm.checkStates()-»
68     «ENDIF-»
69     «ENDFOREACH-»
70
71     «IF type.name == "void"-»
72     «ELSE-»
73     return («type.name») response [0];
74     «ENDIF-»
75     }
76     «ENDIF-»
77 «ENDDEFINE»

```

B.5 Client Template file

```

1 «IMPORT UP4WS»
2
3 «EXTENSION templates::MyExtensions»
4
5 «REM»-----«ENDREM»
6

```

```

7 «DEFINE clientRoot FOR UP4WS::Client»
8   «EXPAND createClientImpl–»
9 «ENDDEFINE»
10
11 «REM»-----«ENDREM»
12
13 «DEFINE createClientImpl FOR UP4WS::Client»
14
15   «FILE name.toFirstUpper()+”.java” myClientOutlet–»
16   package «packageName(this)–»;
17
18   «EXPAND CommonTemplates::createImport FOREACH usedTypes()–»
19
20   import javax.xml.namespace.QName;
21
22   import org.apache.axis2.AxisFault;
23   import org.apache.axis2.addressing.EndpointReference;
24   import org.apache.axis2.client.Options;
25   import org.apache.axis2.rpc.client.RPCServiceClient;
26
27   «visibility» class «name.toFirstUpper()–» {
28
29   public static void main(String[] args1) throws AxisFault {
30
31     RPCServiceClient serviceClient = new RPCServiceClient();
32
33     Options options = serviceClient.getOptions();
34     «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
35     (UP4WS::WebService) AS ws–»
36     EndpointReference targetEPR =
37     new EndpointReference(”http://localhost:8080/axis2/services/«ws.name”);
38     «ENDFOREACH–»
39     options.setTo(targetEPR);
40     «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
41     (UP4WS::ProxyImplementation) AS c–»
42     «c.name.toFirstUpper()–» client = new «c.name.toFirstUpper()–»();
43     «ENDFOREACH–»
44
45     «EXPAND createException
46     FOREACH eAllContents.typeSelect(UP4WS::ClientMain)–»
47     }
48   }
49   «ENDFILE»
50 «ENDDEFINE»
51
52 «REM»-----«ENDREM»
53
54 «DEFINE createException FOR UP4WS::ClientMain»
55 try{
56
57   «FOREACH ((uml::Model) this.eRootContainer).eAllContents.typeSelect
58   (UP4WS::OperationStateMachine) AS wssm–»
59   «IF name+”In”+owner.name == wssm.name–»
60   «EXPAND CommonTemplates::implementState(wssm) FOREACH
61   wssm.eAllContents.typeSelect(UP4WS::OperationState)–»
62   «ENDIF»
63   «ENDFOREACH–»
64
65 }catch (Exception e){

```

```

66     System.out.println(e);
67     }
68 «ENDDEFINE»

```

B.6 MyExtensions Xtend File

```

1  import uml;
2  import Naming;
3  import util;
4
5  extension org::eclipse::xtend::util::stdlib::io reexport;
6  extension org::eclipse::xtend::util::stdlib::counter reexport;
7  extension org::eclipse::xtend::util::stdlib::collections reexport;
8
9
10 String getVisibility(uml::Operation o): o.visibility.toString();
11
12 String constantName(NamedElement this): name.toUpperCase();
13
14 String constantName(State s): s.name.toUpperCase();
15
16 String eventsEnumName(StateMachine this) : name.toFirstUpper()+"Events";
17
18 String statesEnumName(StateMachine this) : name.toFirstUpper()+"States";
19
20 String eventsEnumFileName(StateMachine this) : eventsEnumName()+".java";
21
22 String statesEnumFileName(StateMachine this) : statesEnumName()+".java";
23
24 String stateId(Vertex this, StateMachine m): m.statesEnumName()+"."+constantName();
25
26 String eventId(Event this, StateMachine m): m.eventsEnumName()+"."+constantName();
27
28 cached Collection[State] states (StateMachine this) :
29     region.subvertex.typeSelect(State).sortBy(s|s.name);
30
31 Collection[Transition] transitions (StateMachine this) : region.transition;
32
33 Collection[Transition] outTransitionsWithEventTrigger (State this) :
34     container.stateMachine.transitions().select(t|t.source==this &&
35         t.trigger!=null && !t.trigger.event.contains(null));
36
37
38 Collection[Event] events (StateMachine this) :
39     transitions().trigger.remove(null).event.remove(null).toSet().select
40     (e|e.name!=null).sortBy(e|e.name).toSet();
41
42 String getOwnerName(uml::StateMachine this):
43     ((uml::NamedElement)owner).name;
44
45 List[uml::State] checkStates(uml::StateMachine this):
46     region.ownedMember.typeSelect(uml::State).reject(e|uml::FinalState.isInstance(e));
47
48 int count (uml::Operation o) : o.ownedParameter.sortBy(e|e.name).size;
49
50 String value (uml::Property p): p.getDefault();
51
52 List[uml::Parameter] getParameters(uml::Operation this):

```

```

53     ownedParameter.reject(e|e.direction.toString().toLowerCase().endsWith("return"));
54
55     private cached List[String] umlTypes() : {"Integer","String", "Boolean"};
56     private cached List[String] javaTypes(): {"int", "String", "boolean"};
57
58     String toJava(String this):
59         umlTypes().contains(this) ? javaTypes().get(umlTypes().indexOf(this))
60                                     : this;
61
62     String packageName(uml::Class this):
63         this.getNearestPackage() == null ? ""
64             : this.getNearestPackage().packageName();
65
66     String packageName(uml::Package this):
67         this.owner == null ? name
68             : (owner.packageName().length>0 ? owner.packageName()+ "."+name : name);
69
70     create Set[uml::Type] usedTypes(uml::Class cls):
71         addAll(cls.ownedOperation.ownedParameter.type.typeSelect(uml::Class)) ->
72         removeDuplicatesByName();
73
74     create Set[uml::Type] usedInterfaceTypes(uml::Interface i):
75         addAll(i.ownedOperation.ownedParameter.type.typeSelect(uml::Interface)) ->
76         removeDuplicatesByName();

```


Appendix C

Generated Java Source Code

C.1 Java Classes in the Service Provider Side

C.1.1 Java Classes in data Folder

C.1.1.1 Book.java

```
1 package sample.webservice.data;
2
3 import java.util.*;
4 public class Book {
5
6     private String title;
7
8     private String author;
9
10    private String isbn;
11
12    private BookStatusStates currentState = BookStatusStates.AVAILABLE;
13
14    public void setCurrentState(BookStatusStates currentState) {
15        this.currentState = currentState;
16    }
17    public BookStatusStates getCurrentState() {
18
19        return this.currentState;
20    }
21
22    public void setTitle(String title) {
23        this.title = title;
24    }
25    public String getTitle() {
26
27        return this.title;
28    }
29
30    public void setAuthor(String author) {
31        this.author = author;
```

```
32 }
33 public String getAuthor() {
34
35     return this.author;
36 }
37
38 public void setIsbn(String isbn) {
39     this.isbn = isbn;
40 }
41 public String getIsbn() {
42
43     return this.isbn;
44 }
45
46
47 public String getBookInfo() {
48
49     return new String("Title : " + this.getTitle() + " Isbn : "
50         + this.getIsbn() + " Author : " + this.getAuthor()
51         + "status : " + this.getCurrentState());
52 }
53
54 public void handleEvent(BookStatusEvents event) {
55     switch (currentState) {
56         case AVAILABLE :
57             if (event == BookStatusEvents.LENDBOOK) {
58
59                 currentState = BookStatusStates.ONLOAN;
60
61                 break;
62             }
63             throw new IllegalStateException("Cannot handle event " + event
64                 + " for state " + currentState);
65
66         case NOTAVAILABLE :
67             if (event == BookStatusEvents.ADDBOOK) {
68
69                 currentState = BookStatusStates.AVAILABLE;
70
71                 break;
72             }
73             throw new IllegalStateException("Cannot handle event " + event
74                 + " for state " + currentState);
75
76         case ONLOAN :
77             if (event == BookStatusEvents.RETURNBOOK) {
78
79                 currentState = BookStatusStates.AVAILABLE;
80
81                 break;
82             }
83             throw new IllegalStateException("Cannot handle event " + event
84                 + " for state " + currentState);
85     }
86 }
87 }
```


C.1.1.2 BookList.java

```
1 package sample.webservice.data;
2
3 import sample.webservice.data.*;
4
5 import java.util.*;
6 public class BookList {
7
8     private HashMap bookTable = new HashMap();
9
10    public void setBookTable(HashMap bookTable) {
11        this.bookTable = bookTable;
12    }
13    public HashMap getBookTable() {
14
15        return this.bookTable;
16    }
17
18    public void addBook(Book book) {
19
20        bookTable.put(book.getIsbn(), book);
21    }
22 }
23
24 public Book getBook(String isbn) {
25
26     return (Book) bookTable.get(isbn);
27 }
28 }
```

C.1.1.3 BookStatusEvents.java

```
1 package sample.webservice.data;
2
3 public enum BookStatusEvents {
4
5     ADDBOOK, LENDBOOK, RETURNBOOK
6 }
```

C.1.1.4 BookStatusStates.java

```
1 package sample.webservice.data;
2
3 public enum BookStatusStates {
4
5     AVAILABLE, FINALSTATE, NOTAVAILABLE, ONLOAN
6
7 }
```

C.1.2 Java Classes in service Folder

C.1.2.1 LibrarySystemService.java

```
1 package sample.webservice.service;
2
3 import sample.webservice.data.*;
4
```

```
5 public class LibrarySystemService {
6
7     private BookList allBookList = new BookList();
8
9     public BookStatusStates getBookStatus(String isbn) {
10
11         Book book = allBookList.getBook(isbn);
12
13         return book.getCurrentState();
14     }
15
16     public void setAllBookList(BookList allBookList) {
17         this.allBookList = allBookList;
18     }
19     public BookList getAllBookList() {
20
21         return this.allBookList;
22     }
23
24     public String lendBook(String isbn) {
25
26         String bookInfo = new String();
27         Book book = allBookList.getBook(isbn);
28
29         if (book == null) {
30             System.out.println("this book is not found...");
31             return bookInfo;
32         } else {
33             book.handleEvent(BookStatusEvents.LENDBOOK);
34         }
35         return bookInfo = book.getBookInfo();
36     }
37
38     public void addBook(String author, String isbn, String title) {
39
40         Book book = new Book();
41
42         book.setCurrentState(BookStatusStates.AVAILABLE);
43         book.setTitle(title);
44         book.setIsbn(isbn);
45         book.setAuthor(author);
46         allBookList.addBook(book);
47     }
48 }
49
50 public void returnBook(String isbn) {
51
52     Book book = allBookList.getBook(isbn);
53
54     if (book == null) {
55         System.out.println("this book is not found...");
56     } else {
57         book.handleEvent(BookStatusEvents.RETURNBOOK);
58     }
59 }
60 }
61
62 public Book getBook(String isbn) {
63
```

```
64     return (Book) allBookList.getBook(isbn);
65 }
66 }
```

C.2 Java Classes in the Service Client Side

C.2.1 Java Classes in client Folder

C.2.1.1 LibrarySystemServiceClient.java

```
1  package sample.webservice.client;
2
3  import sample.webservice.data.*;
4
5  import javax.xml.namespace.QName;
6  import org.apache.axis2.AxisFault;
7  import org.apache.axis2.addressing.EndpointReference;
8  import org.apache.axis2.client.Options;
9  import org.apache.axis2.rpc.client.RPCServiceClient;
10
11 public class LibrarySystemServiceClient {
12
13     public void addBook(String author, String isbn, String title,
14         RPCServiceClient rpcClient) throws AxisFault {
15
16         QName opGet = new QName("http://service.webservice.sample", "addBook");
17         rpcClient.getOptions().setAction("urn:addBook");
18
19         Object[] args = new Object[3];
20
21         args[0] = author;
22         args[1] = isbn;
23         args[2] = title;
24
25         rpcClient.invokeRobust(opGet, args);
26
27     }
28
29     public void returnBook(String isbn, RPCServiceClient rpcClient)
30         throws AxisFault {
31
32         QName opGet = new QName("http://service.webservice.sample",
33             "returnBook");
34         rpcClient.getOptions().setAction("urn:returnBook");
35
36         Object[] args = new Object[1];
37
38         args[0] = isbn;
39
40         rpcClient.invokeRobust(opGet, args);
41
42     }
43
44     public String lendBook(String isbn, RPCServiceClient rpcClient)
45         throws AxisFault {
46
47         QName opGet = new QName("http://service.webservice.sample", "lendBook");
48         rpcClient.getOptions().setAction("urn:lendBook");
```

```

49
50     Object[] args = new Object[1];
51
52     args[0] = isbn;
53
54     Class[] returnTypes = new Class[]{String.class};
55     Object[] response = rpcClient.invokeBlocking(opGet, args, returnTypes);
56
57     return (String) response[0];
58 }
59
60 public Book getBook(String isbn, RPCServiceClient rpcClient)
61     throws AxisFault {
62
63     QName opGet = new QName("http://service.webservice.sample", "getBook");
64     rpcClient.getOptions().setAction("urn:getBook");
65
66     Object[] args = new Object[1];
67
68     args[0] = isbn;
69
70     Class[] returnTypes = new Class[]{Book.class};
71     Object[] response = rpcClient.invokeBlocking(opGet, args, returnTypes);
72
73     Book book = (Book) response[0];
74
75     if (book != null) {
76         System.out.println(book.getBookInfo());
77     } else {
78         System.out.println("book is not found ...");
79     }
80
81     return (Book) response[0];
82 }
83 }

```

C.2.1.2 LibrarySystemServiceClientMain.java

```

1  package sample.webservice.client;
2
3  import javax.xml.namespace.QName;
4  import org.apache.axis2.AxisFault;
5  import org.apache.axis2.addressing.EndpointReference;
6  import org.apache.axis2.client.Options;
7  import org.apache.axis2.rpc.client.RPCServiceClient;
8
9  public class LibrarySystemServiceClientMain {
10
11     public static void main(String[] args1) throws AxisFault {
12
13         System.out.println("start main");
14         RPCServiceClient serviceClient = new RPCServiceClient();
15         System.out.println("start rpc client");
16         Options options = serviceClient.getOptions();
17         System.out.println("start epr");
18         EndpointReference targetEPR = new EndpointReference(
19             "http://localhost:8080/axis2/services/LibrarySystemService");
20         System.out.println("start srt to option");
21         options.setTo(targetEPR);

```

```
22 LibrarySystemServiceClient client = new LibrarySystemServiceClient();
23 System.out.println("start lib client");
24
25 try {
26     client.addBook("My Author", "730076", "UML2", serviceClient);
27
28     String bookInfo = client.lendBook("730076", serviceClient);
29     System.out.println(bookInfo);
30
31     client.returnBook("730076", serviceClient);
32
33 } catch (Exception e) {
34     System.out.println(e);
35 }
36 }
37 }
38 }
```


Appendix D

Transformation Rules for Generating Configuration Files

D.1 XmlFiles Template File

```
1 «REM» XmlFiles Template File «ENDREM»
2
3 «DEFINE xmlRoot FOR uml::Model»
4
5 «EXPAND createBuildXML
6   FOREACH this.eAllContents.typeSelect(UP4WS::WebService)»
7 «EXPAND createServicesXML
8   FOREACH this.eAllContents.typeSelect(UP4WS::WebService)»
9
10 «ENDDEFINE»
11
12 «REM»-----«ENDREM»
13 «DEFINE createBuildXML FOR UP4WS::WebService»
14 «FILE "build.xml" buildXML-»
15   «REM»Project name holding the name of the Web service«ENDREM»
16   <project name="«name»"
17
18     basedir="."
19
20     default="generate.service">
21     «REM»The name of the Web service«ENDREM»
22     <property name="service.name" value="«name»"/>
23     <property name="dest.dir" value="build"/>
24     <property name="dest.dir.classes" value="${dest.dir}/${service.name}"/>
25     <property name="dest.dir.lib" value="${dest.dir}/lib"/>
26     <property name="axis2.home" value="../.."/>
27     <property name="repository.path" value="${axis2.home}/repository"/>
28
29     <path id="build.class.path">
30       <fileset dir="${axis2.home}/lib">
31         <include name="*.jar"/>
32       </fileset>
```

```

33 </path>
34
35 <path id="client.class.path">
36   <fileset dir="${axis2.home}/lib">
37     <include name="*.jar"/>
38   </fileset>
39   <fileset dir="${dest.dir.lib}">
40     <include name="*.jar"/>
41   </fileset>
42
43 </path>
44 <target name="clean">
45   <delete dir="${dest.dir}"/>
46
47   <delete dir="src" includes="sample/webservice/stub/**"/>
48 </target>
49
50 <target name="prepare">
51   <mkdir dir="${dest.dir}"/>
52   <mkdir dir="${dest.dir}/lib"/>
53   <mkdir dir="${dest.dir.classes}"/>
54   <mkdir dir="${dest.dir.classes}/META-INF"/>
55 </target>
56
57
58 <target depends="clean,prepare" name="generate.service">
59
60   <copy file="src/META-INF/services.xml" overwrite="true" tofile=
61 "${dest.dir.classes}/META-INF/services.xml"/>
62
63
64
65   <javac destdir=
66 "${dest.dir.classes}" includes="
67 sample/webservice/service/**,sample/webservice/data/**" srcdir="src">
68   <classpath refid="build.class.path"/>
69 </javac>
70
71   <jar basedir="${dest.dir.classes}" destfile="${dest.dir}/${service.name}.aar"/>
72
73   <copy file="${dest.dir}/${service.name}.aar" overwrite="true"
74 tofile="${repository.path}/services/${service.name}.aar"/>
75
76 </target>
77
78 <target depends="clean,prepare" name="rpc.client">
79
80   <antcall target="rpc.client.compile"/>
81
82   <antcall target="rpc.client.jar"/>
83
84   <antcall target="rpc.client.run"/>
85
86 </target>
87
88 <target name="rpc.client.compile">
89   <javac destdir="${dest.dir.classes}" includes="
90 sample/webservice/client/**,sample/webservice/data/**" srcdir="src">

```



```

92     <classpath refid="build.class.path"/>
93     </javac>
94 </target>
95
96 <target name="rpc.client.jar">
97     <jar basedir=
98     "${dest.dir.classes}" destfile="${dest.dir.lib}/rpc-client.jar" includes="
99     sample/webservice/client/**,sample/webservice/data/**"/>
100 </target>
101
102 «REM»get the name of the Web service«ENDREM»
103 <target name="rpc.client.run">
104     <java classname="sample.webservice.client.«name»Client">
105         <classpath refid="client.class.path"/>
106     </java>
107     <java classname="sample.webservice.client.«name»ClientMain">
108         <classpath refid="client.class.path"/>
109     </java>
110 </target>
111 </project>
112
113 «ENDFILE»
114 «ENDDEFINE»
115
116 «REM»-----«ENDREM»
117 «DEFINE createServicesXML FOR UP4WS::WebService»
118     «FILE "services.xml" servicesXML-»
119     «REM»get the name of the Web service«ENDREM»
120     <service name="«name»" scope="application">
121         «REM»get the name of the comment attached to the Web service Java class«ENDREM»
122         <description>
123             «ownedComment.get(0).body»
124         </description>
125         <messageReceivers>
126             <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
127                 class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
128             <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
129                 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
130         </messageReceivers>
131         «REM»get the name of the Web service«ENDREM»
132         <parameter name="ServiceClass">sample.webservice.service.«name»</parameter>
133
134     </service>
135     «ENDFILE»
136 «ENDDEFINE»
137
138 «REM»-----«ENDREM»

```

D.2 build.xml File

```

1 <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2 <project basedir="." default="generate.service" name="LibrarySystemService">
3
4     <property name="service.name" value="LibrarySystemService"/>
5     <property name="dest.dir" value="build"/>
6     <property name="dest.dir.classes" value="${dest.dir}/${service.name}"/>
7     <property name="dest.dir.lib" value="${dest.dir}/lib"/>
8     <property name="axis2.home" value="../../"/>

```

```

 9  <property name="repository.path" value="${axis2.home}/repository"/>
10
11  <path id="build.class.path">
12    <fileset dir="${axis2.home}/lib">
13      <include name="*.jar"/>
14    </fileset>
15  </path>
16
17  <path id="client.class.path">
18    <fileset dir="${axis2.home}/lib">
19      <include name="*.jar"/>
20    </fileset>
21    <fileset dir="${dest.dir.lib}">
22      <include name="*.jar"/>
23    </fileset>
24
25  </path>
26  <target name="clean">
27    <delete dir="${dest.dir}"/>
28
29    <delete dir="src" includes="sample/webservice/stub/**"/>
30  </target>
31
32  <target name="prepare">
33    <mkdir dir="${dest.dir}"/>
34    <mkdir dir="${dest.dir}/lib"/>
35    <mkdir dir="${dest.dir.classes}"/>
36    <mkdir dir="${dest.dir.classes}/META-INF"/>
37  </target>
38
39
40  <target depends="clean,prepare" name="generate.service">
41
42
43    <copy file="src/META-INF/services.xml" overwrite="true"
44  tofile="${dest.dir.classes}/META-INF/services.xml"/>
45
46
47    <javac destdir="${dest.dir.classes}"
48  includes="sample/webservice/service/**,sample/webservice/data/**" srcdir="src">
49
50      <classpath refid="build.class.path"/>
51    </javac>
52
53    <jar basedir="${dest.dir.classes}"
54  destfile="${dest.dir}/${service.name}.aar"/>
55
56    <copy file="${dest.dir}/${service.name}.aar" overwrite="true"
57  tofile="${repository.path}/services/${service.name}.aar"/>
58
59  </target>
60
61  <target depends="clean,prepare" name="rpc.client">
62
63    <antcall target="rpc.client.compile"/>
64
65    <antcall target="rpc.client.jar"/>
66
67    <antcall target="rpc.client.run"/>

```

```

68
69 </target>
70
71 <target name="rpc.client.compile">
72   <javac destdir="${dest.dir.classes}"
73 includes="sample/webservice/client/**,sample/webservice/data/**" srcdir="src">
74
75     <classpath refid="build.class.path"/>
76   </javac>
77 </target>
78
79 <target name="rpc.client.jar">
80   <jar basedir="${dest.dir.classes}"
81 destfile="${dest.dir.lib}/rpc-client.jar"
82 includes="sample/webservice/client/**,sample/webservice/data/**"/>
83 </target>
84
85
86 <target name="rpc.client.run">
87   <java classname="sample.webservice.client.LibrarySystemServiceClient">
88     <classpath refid="client.class.path"/>
89   </java>
90   <java classname="sample.webservice.client.LibrarySystemServiceClientMain">
91     <classpath refid="client.class.path"/>
92   </java>
93 </target>
94 </project>

```

D.3 services.xml File

```

1 <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2 <service name="LibrarySystemService" scope="application">
3
4   <description>
5     Library System Web Service
6   </description>
7   <messageReceivers>
8     <messageReceiver
9 class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"
10 mep="http://www.w3.org/2004/08/wsdl/in-only"/>
11
12     <messageReceiver
13 class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"
14 mep="http://www.w3.org/2004/08/wsdl/in-out"/>
15   </messageReceivers>
16
17   <parameter name="ServiceClass">sample.webservice.service.LibrarySystemService
18 </parameter>
19
20 </service>

```


Appendix E

Transformation Rules for Generating the README File

E.1 README Template File

```
1 «IMPORT UP4WS»
2
3 «DEFINE readmeTmpl FOR UP4WS::WebService»
4
5     «FILE "README.txt"–»
6
7 Introduction
8 =====
9 This case study («name» Web Service) has been implemented to validate the proposed
10 UML profile for Web services and the code generation process.
11 The case study shows a complete working example that represents the modeling of Web services
12 with UML, and the model transformation and code generation of source code and configuration
13 files for the relevant platform.
14 In this case study, both perspectives of service provider and service client has been introduced.
15
16 Web Services using Apache Axis2
17 =====
18 This program contains the source code for the «name» Web service.
19 This source code is resulted from a UML model developed using a UML Tool,
20 and transformed into a java code using Xpand language together with some Xtend
21 functions in Eclipse.
22
23 Prerequisites and Environment Settings
24 =====
25 1. Windows as an Operating System (The case study has been implemented on
26 Windows XP and Vista)
27 2. Apache Ant 1.6.2 or later
28 3. Apache Axis2
29 4. Apache Tomcat
30 5. Java
31
32 How to deploy the service on the Application Server?
```

```

33 =====
34 1. Compile the server side (service provider side) classes with ANT.
35 2. Put the «name».aar inside the services folder in Tomcat
36 (... \apache-tomcat\webapps\axis2\WEB-INF\services)
37 3. Initiate the tomcat server by typing the relevant command e.g. tomcat.exe or startup.bat
38 from inside the bin folder.
39 4. Make sure that the service is correctly deployed by typing
40 http://localhost:8080/axis2/services/«name»?wsdl in your browser
41
42 Compile the Client side Classes by typing the following commands:
43 =====
44 First go to the root where the build.xml file resides:
45
46 javac -sourcepath src\sample\webservice\client -classpath build\«name»
47 -extdirs "C:\ApacheInstallation\axis2-1.5\lib"
48 -d build\«name» src\sample\webservice\client\«name»Client.java
49
50 javac -sourcepath src\sample\webservice\client -classpath build\«name»
51 -extdirs "C:\ApacheInstallation\axis2-1.5\lib"
52 -d build\«name» src\sample\webservice\rpcclient\«name»ClientMain.java
53
54 Running the Client Side classes by typing the following commands:
55 =====
56 First go to the root which ends with ... \build\«name»
57 N.B. The name of the folder that contains your Web service files
58 must hold the same name of your Web service.
59
60 java -Djava.ext.dirs="C:\ApacheInstallation\axis2-1.5\lib"
61 sample.webservice.client.«name»ClientMain
62
63 Help
64 ===
65 Please contact (wdahman@informatik.uni-goettingen.de or arrivalsw@yahoo.com)
66 if you have any troubles running the Web service.
67
68 «ENDFILE»
69
70 «ENDDEFINE»

```

E.2 README.txt File

```

1
2 Introduction
3 =====
4 This case study (LibrarySystemService Web Service) has been implemented to validate
5 the proposed UML profile for Web services and the code generation process.
6 The case study shows a complete working example that represents the modeling
7 of Web services with UML, and the model transformation and code generation of
8 source code and configuration files for the relevant platform.
9 In this case study, both perspectives of service provider and service client has been introduced.
10
11 Web Services using Apache Axis2
12 =====
13 This program contains the source code for the LibrarySystemService Web service.
14 This source code is resulted from a UML model developed using a UML Tool,
15 and transformed into a java code using Xpand language together with some Xtend
16 functions in Eclipse.
17

```

```

18 Prerequisites and Environment Settings
19 =====
20 1. Windows as an Operating System (The case study has been implemented on
21 Windows XP and Vista)
22 2. Apache Ant 1.6.2 or later
23 3. Apache Axis2
24 4. Apache Tomcat
25 5. Java
26
27
28 How to deploy the service on the Application Server?
29 =====
30 1. Compile the server side (service provider side) classes with ANT.
31 2. Put the LibrarySystemService.aar inside the services folder in Tomcat
32 (...apache-tomcat\webapps\axis2\WEB-INF\services)
33 3. Initiate the tomcat server by typing the relevant command e.g. tomcat.exe or startup.bat
34 from inside the bin folder.
35 4. Make sure that the service is correctly deployed by typing
36 http://localhost:8080/axis2/services/LibrarySystemService?wsdl in your browser
37
38 Compile the Client side Classes by typing the following commands:
39 =====
40
41 First go to the root where the build.xml file resides:
42
43 javac -sourcepath src\sample\webservice\client -classpath build\LibrarySystemService
44 -extdirs "C:\ApacheInstallation\axis2-1.5\lib" -d build\LibrarySystemService
45 src\sample\webservice\client\LibrarySystemServiceClient.java
46
47 javac -sourcepath src\sample\webservice\client -classpath build\LibrarySystemService
48 -extdirs "C:\ApacheInstallation\axis2-1.5\lib" -d build\LibrarySystemService
49 src\sample\webservice\rpcclient\LibrarySystemServiceClientMain.java
50
51
52 Running the Client Side classes by typing the following commands:
53 =====
54
55 First go to the root which ends with ...build\LibrarySystemService
56 N.B. The name of the folder that contains your Web service files must hold the same name of
57 your Web service.
58
59 java -Djava.ext.dirs="C:\ApacheInstallation\axis2-1.5\lib"
60 sample.webservice.client.LibrarySystemServiceClientMain
61
62
63 Help
64 ===
65 Please contact (wdahman@informatik.uni-goettingen.de or arrivalsw@yahoo.com)
66 if you have any troubles running the Web service.

```


Appendix F

Executing the Web Service

F.1 Web Service Engine

The Web service engine is a collection of pieces of software and hardware for the implementation and deployment of Web services applications. One of the popular examples of Web service engines is the Apache Axis2. The Apache Axis2 has been adopted during this research to deploy and implement Web services. The reason behind this decision is the fact that it is an open source Web service engine, by which the flexibility, and scalability are guaranteed. Furthermore, it has a large scale of quality attributes with respect to software applications in general and Web services in particular. For example, the Apache Axis2 supports flexibility, stability, composition and extensibility of Web services. In addition, it allows the addition of different add-ons for security, and new standards to increase reliability [AA2c]. The Apache Axis2 Web services engine has two different Web services implementations:

F.1.1 Apache Axis2/Java

As the name indicates this type is using Java as a programming language for the implementation of Web services. The Apache Axis2/Java engine [AA2c] is used in this research, since the Java Web services community has a large involvement in the development of Web services. This option has the advantage that the examples of Web services implemented in Java can be used in different Web services engines without massive changes in the source code.

F.1.2 Apache Axis2/C

Apache Axis2/C [AA2a] is another implementation of the Apache Axis2 Web service engine. It utilises the C programming language for the implementation

of Web services. Apache Axis2/C supports different types of WS-standards like WS-Addressing [WSA] and WS-ReliableMessaging [WSR].

F.2 Setting the Environments

In order to run the Web service, the different environments proposed in this thesis have to be established, and configured. The Web service in this thesis is designed to run on the following platform characteristics.

F.2.1 Web Service Engine

The Apache Axis2/Java is the Web service engine, which has been used for the implementation and deployment of the Web service. Some artifacts belong to the Apache Axis2 appear in the Java source code at the service client side. The installation and configuration of the Apache Axis2 are at [AA2b].

F.2.2 Application Server

Although the Apache Axis2 has its own standalone application server, the Apache Tomcat has been used as an application server to host the Web service. It is an open source and can be easily configured with Apache Web service engine. The installation and configuration of the Apache Tomcat are at [ATO].

F.2.3 Operating System

The operating system, on which the Web service has been executed, is Windows. Environment variables have to be set and defined before executing the Web service. Instructions on how to set and define these variables are at [AA2b, ATO]

F.2.4 Additional Considerations

The following points must be taken into consideration before running the Web service.

- Export your model from the modelling tool to the generator tool in EMF UML2 XMI.
- Locate your exported model inside the same Xpand project folder as specified in the workflow file.
- Locate the output folder and its contents inside the sample folder, which is found inside the folder produced when unzipping the Apache Axis2 distribution.

- Follow the instructions found in the *README.txt* (see Listing 3.26) file, which is automatically generated together with the output files.

List of Symbols and Abbreviations

BPEL4WS Business Process Execution Language for Web Services

CIM Computation Independent Model

CORBA Common Object Request Broker Architecture

CRC Class, Responsibilities, Collaborations

CWM Common Warehouse Model

EMF Eclipse Modeling Framework

HTTP Hyper Text Transfer Protocol

IBM International Business Machines

ISBN International Serial Book Number

M2C Model-to-Code

M2M Model-to-Model

M2T Model-to-Text

MDA Model Driven Architecture

MOF Meta Object Facility

MWE Modeling Workflow Engine

OCL Object Constraint Language

OGSI Open Grid Services Infrastructure

OMG Object Management Group

OMT	Object Modeling Technique
PIM	Platform Independent Model
PM	Platform Model
PSM	Platform Specific Model
RFP	Request For Proposal
RUP	Rational Unified Process
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
T2M	Text-to-Model
T2T	Text-to-Text
UDDI	Universal, Description, Discovery, and Integration
UML	Unified Modeling Language
UML2	Unified Modeling Language v.2.x
UP4WS	UML Profile for Web Services
W3C	World Wide Web Consortium
WSC	Web Service Composition
WSDL	Web Service Description Language
WSDM	Web Services Development Model
WSRF	WS-Resource Framework
XMI	XML Metadata Interchange
XML	Xtensible Mark-up Language
xUML	Executable UML

List of Figures

1.1	Thesis Structure Overview	6
2.1	Roles in the Service Oriented Architecture (SOA)	10
2.2	Sample Use Case Diagram	14
2.3	Sample State Machine Diagram	15
2.4	Sample Class Diagram	16
2.5	UML Metamodel Layers (Example)	17
2.6	UML <i>WidthAndLength</i> Profile	18
2.7	A UML View for MDA Models	20
2.8	Protected Regions	24
2.9	Book-Publisher Class Diagram	28
3.1	Basic Extensions and Their Usage	36
3.2	«ObjectStateMachine» Stereotype Implementation	41
3.3	«DataBehavior» Stereotype	44
3.4	«OperationStateMachine» Stereotype Implementation	45
3.5	«Initializable» Stereotype Example	48
3.6	«Enumerizable» Stereotype Example	49
3.7	Model Transformation and Code Generation Process	49
4.1	Web Services Development Model Overview	69
4.2	Sample Use Case Diagrams	72
4.3	Class Diagram for Requirements Analysis	73
4.4	A Class Diagram in Architecture Design with UP4WS Stereotypes	76
5.1	Case Study Implementation	82
5.2	Requirements Analysis: Library System Web Service Use Cases	84
5.3	Library System Web Services Initial Architecture	85
5.4	Class Diagram at Service Provider Side	88
5.5	Book Status	89

5.6	Code Generator Successful Execution	95
5.7	LibrarySystemService Folder After Generator Execution	96
A.1	Class Diagram at Service Client Side	109
A.2	getBookInLibrarySystemServiceClient	109
A.3	mainInLibrarySystemServiceClientMain	110
A.4	Class Diagram at Service Provider Side	110
A.5	BookStatus	111
A.6	addBookInBookList	111
A.7	getBookInBookList	111
A.8	getBookInfoInBook	111
A.9	addBookInLibrarySystemService	112
A.10	getBookStatusInLibrarySystemService	112
A.11	lendBookInLibrarySystemService	112
A.12	returnBookInLibrarySystemService	112

Listings

2.1	Protected Regions in Xpand	24
2.2	Template Definition	25
2.3	Import Statement	26
2.4	Extension Statement	26
2.5	FILE Block Definition	26
2.6	FOREACH Block	27
2.7	EXPAND Statement A	27
2.8	EXPAND Statement B	27
2.9	IF Statement	27
2.10	REM Block	28
2.11	Xpand in Example	28
2.12	Java Class Book	30
2.13	Java Class Publisher	30
2.14	Sample FOREACH in Xpand	31
2.15	Xtend Sample Operation	31
2.16	Using Extensions in Xpand	31
3.1	createImport Template	50
3.2	createDataElement Template	51
3.3	Enumerizable Template	51
3.4	gettersAndSetters Template	52
3.5	methodTmpl Template	52
3.6	Events Template	53
3.7	States Template	53
3.8	implementState Template	53
3.9	serviceRoot Template	54
3.10	createWebService Template	55
3.11	proxyMethodServiceTmpl Template	55
3.12	dataContainerRoot Template	56
3.13	createDataContainer Template	57
3.14	ObjectBehavior Template	58

3.15	executeTransitionForHandleEvent Template	58
3.16	illegalTransitionHandler Template	59
3.17	proxyRoot Template	59
3.18	createProxyImpl Template	59
3.19	proxyMethodClientTmpl Template	60
3.20	clientRoot	61
3.21	createClientImpl Template	61
3.22	createException Template	62
3.23	xmlRoot Template	64
3.24	createBuildXML Template	64
3.25	createServicesXML Template	66
3.26	README Template	67
5.1	MWE File for the Library System Web Service	89
5.2	Book Object Java Source Code	91
5.3	<i>services.xml</i> File for the Library System Web Service	93
5.4	Library System Web Service <i>README.txt</i> File	94

List of Tables

3.1	Mapping Between Template Files and Stereotypes	50
4.1	CRC Card for Service Object	72
5.1	CRC Card for Library System Service Object	83
5.2	CRC Card for Book Object	84

Curriculum Vitae

Wafi Abed Zaidan Dahman

Persönliche Daten

Geburt 19. April 1978 in Gaza - Palästina

Staatsangehörigkeit Palästinensisch

Wissenschaftlicher Werdegang

1984-1990 Grundschule, Gaza "A" School- Palästina

1990-1993 Al-Zaitoun School, Palästina

1993-1996 Al-Karmel Secondary School - Palästina

Abschluss:

06/1996 General Secondary Education Certificate
(Scientific Stream)

1996-1998 Gaza Training Center/College (GTC)

Abschluss:

Business and Office Practice (2J. Diploma)

1998-2001 Islamic University - Palästina

Faculty of Commerce - Accounting

Abschluss:

02/2001 Bachelor in Commerce - Accounting

2001-2003 The Arab Academy for Banking and Financial Sciences

Computer Science Department

Abschluss:

10/2003 Master in Computer Information Systems

seit 2006 Georg-August-Universität Göttingen - Germany

Doktorand - DAAD Stipendiat

am Institut für Informatik

Forschungsgruppe für Softwaretechnik für Verteilte Systeme

10 Juni 2010