



Norwegian University of
Science and Technology

Model Driven Development of Web Application with SPACE Method and Tool-suit

Jinat Rehana

Master in Security and Mobile Computing

Submission date: June 2010

Supervisor: Peter Herrmann, ITEM

Co-supervisor: Simon Han, ITEM

Problem Description

AJAX technology is a very hot client-side technology for building interactive web applications. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. Right now, there are already many matured AJAX frameworks. Among them Google Web Toolkit (GWT for short, <http://code.google.com/webtoolkit/>) is developed by google. GWT is pure open source and handy for use in eclipse environment. As google technology are getting more and more popular, many software companies are also taking GWT as their framework for developing their software product. We want to explore into the possibility of providing building blocks for web application (AJAX application) development integrating GWT with SPACE method. SPACE is a model driven method for rapid creation of services from reusable building blocks, developed at NTNU. Arctis is a tool, also developed at NTNU to support SPACE method. Several solutions have been developed on Arctis tools covering several domains like mobile services, embedded systems, home automation, trust management and web services.

Our contribution in this thesis will be to do a case study on web application domain with Arctis, where the underlying technology will be AJAX and GWT (Google Web Toolkit). In order to do that, we will try to build up some reusable building blocks with arctis tool. We will describe a use case scenario to use those building blocks. We will try to implement our specified system and evaluate our design specification based on reusability factor.

Assignment given: 15. January 2010
Supervisor: Peter Herrmann, ITEM

Abstract

Enterprise level software development using traditional software engineering approaches with third-generation programming languages is becoming more challenging and cumbersome task with the increased complexity of products, shortened development cycles and heightened expectations of quality. MDD (Model Driven Development) has been counting as an exciting and magical development approach in the software industry from several years. The idea behind MDD is the separation of business logic of a system from its implementation details expressing problem domain using models. This separation and modeling of problem domain simplify the process of system design as well as increase the longevity of products as new technologies can be adopted easily. With appropriate tool support, MDD shortens the software development life cycle drastically by automating a significant portion of development steps. MDA (Model Driven Architecture) is a framework launched by OMG (Object Management Group) to support MDD. SPACE is an engineering method for rapid creation of services, developed at NTNU (Norwegian University of Science and Technology) which follows MDA framework. Arctis and Ramses are tool suits, also developed at NTNU to support SPACE method. Several solutions have been developed on Arctis tool suit covering several domains like mobile services, embedded systems, home automation, trust management and web services.

This thesis presents a case study on the web application domain with Arctis, where the underlying technologies are AJAX (asynchronous JavaScript and XML), GWT (Google Web Toolkit) framework and Java Servlet. In order to do that, this thesis contributes building up some reusable building blocks with Arctis tool suit. This thesis also describes a use case scenario to use those building blocks. This thesis work tries to implement the specified system and evaluates the resulting work.

Acknowledgements

At first, I would like to express my gratitude to the Almighty for giving me enough strength to carry out this work.

I would like to thank Prof. Peter Herrmann from NTNU and Prof. Tuomas Aura from AYY for supervising me with their proper guidance. Their helpful feedback has forwarded me to the right direction.

I would like to give my special thanks to Fenglin Han, my co-supervisor of this thesis work. His continuous help, support and suggestions have shaped this work.

I would also like to thank my family and my beloved husband Kashif Nizam Khan for their continuous support and inspiration to complete this work.

Finally, thanks to my friends and near ones for inspiring me during this journey.

Trondheim June 30th, 2010.

Jinat Rehana

Abbreviations and Acronyms

MDD	Model Driven Development
MDA	Model Driven Architecture
PIM	Platform Independent Model
PSM	Platform Specific Model
OMG	Object Management Group
UML	Unified Modeling Language
TLA	Temporal Logic of Actions
cTLA	compositional Temporal Logic of Actions
AJAX	Asynchronous JavaScript And XML
GWT	GoogleWeb Toolkit
3GL	Third Generation Programming Language
OO-H Method	Object- Oriented Hypermedia Method
NAD	Navigation Access Diagram
APD	Abstract Presentation Diagram
CLD	Composition Layout Diagram
MOF	MetaObject Facility
UWE	UML based Web Engineering
QVT	Query/View/Transformation
OCL	Object Constraint Language
XMI	XML Metadata Interchange
PIM	Process Information Model
MDE	Model Driven Engineering
CIM	Computation-Independent Model
ESM	External State Machine
XHTML	eXtensible HyperText Markup Language
CSS	Cascading Style Sheet
DOM	Document Object Model
XML	Extensible Markup Language
XSLT	eXtensible Style Sheet Language Transformations
HTML	Hyper Text Markup Language
RIA	Rich Internet Application

JRE	Java Runtime Environment
UI	User Interface
RPC	Remote Procedure Call
ER diagram	Entity Relationship diagram

Contents

Abstract	i
Abbreviations and Acronyms	v
List of Figures	xiii
1 Introduction	1
1.1 Context	1
1.2 Problem Statement	3
1.3 Related Works	3
1.4 Objective	6
1.5 Contribution	7
1.6 Thesis Outline	7
2 Background	9
2.1 MDD (Model Driven Development)	9
2.2 MDA (Model Driven Architecture MDA)	10
2.3 SPACE Method	11
2.3.1 System Specification or Service Composition	11
2.3.2 Formal Analysis and Automated model Checking	20
2.3.3 Model Transformation	21
2.3.4 Code Generation	22
2.4 Arctis and Ramses Tool-Suits to Support SPACE method	22
2.5 AJAX (Asynchronous JavaScript and XML)	24

2.5.1	Working Principle of AJAX	25
2.5.2	What actually AJAX is	26
2.6	Java Servlet	29
2.7	Google Web Toolkit (GWT)	30
2.7.1	GWT Components	31
2.7.2	GWT Modes of Running	32
3	AJAX Integration in Arctis	34
3.1	Client-Server Communication with GWT Framework	35
3.1.1	GWT RPC	36
3.1.2	How GWT RPC Works	41
3.2	Development of the Client-Server Communication Model in Arctis	42
3.3	GWT Code Generator	44
3.3.1	GWT application layout Generation	45
3.3.2	Automatic Generation of the Server Methods' Signatures	46
4	Scenario Description	48
4.1	General Scenario	48
4.2	Online Shopping System as a Scenario	48
4.2.1	Scope of the Example Scenario	49
4.2.2	Use Case Diagram of the Example Scenario	50
4.2.3	Example UI Representing the Example Scenario	50
4.2.4	Example Database	52
5	Development of Building Blocks	56
5.1	System Block of the Online Shopping System	56
5.2	MySQL Building Block	61
5.2.1	ESM of the MySQL Building Block	62
5.3	LogInGUI_Proxy Building Block	63
5.3.1	ESM of the LogInGUI_Proxy Building Block	64

5.4	LogIn Building Block	64
5.4.1	ESM of the LogIn Building Block	66
5.5	ServiceQuery_Proxy Building Block	66
5.5.1	ESM of the ServiceQuery_Proxy Building Block	67
5.6	ServiceQuery Building Block	68
5.6.1	ESM of the ServiceQuery Building Block	69
5.7	AddtoCartGUI_Proxy Building Block	70
5.7.1	ESM of the AddtoCartGUI_Proxy Building Block	71
5.8	AddtoCart Building Block	72
5.8.1	ESM of the AddtoCart Building Block	74
5.9	OrderInfoEnabler_Proxy Building Block	75
5.9.1	ESM of the OrderInfoEnabler_Proxy Building Block	75
5.10	PlacingOrderEnabler Building Block	76
5.10.1	ESM of the PlacingOrderEnabler Building Block	77
5.11	OrderingUI_Proxy Building Block	78
5.11.1	ESM of the OrderingUI_Proxy Building Block	79
5.12	PlacingOrder Building Block	80
5.12.1	ESM of the PlacingOrder Building Block	81
5.13	Our Service Specification in the Perspective of the SPACE Method	82
6	Implementation	85
6.1	Model Checking	85
6.1.1	Analysis on Building Blocks	85
6.1.2	Analysis on the Specified System	89
6.2	Implementation of the Client-Server Communication Model	91
6.3	Implementation of the ServiceQuery Building Block	94
6.4	Implementation of a Test System	96
6.5	Multisession Issue	100
7	Discussion	103

7.1	Evaluation of the Resulting Work	103
7.2	Limitations	105
7.3	Future Work	105
	Bibliography	107

List of Figures

2.1	Software Development Lifecycle with MDA	10
2.2	System Development Cycle in SPACE method (adopted from [43])	12
2.3	Services as Collaborations among Several Components. C depicts system components, S depicts services and dotted rectangle comprises the whole system.	13
2.4	Collaboration of the Mobile Alarm System [43]	14
2.5	System as Hierarchy of Building Blocks	15
2.6	Behavior of the MobileAlarmSystem Collaboration	16
2.7	Internal Behavior of the TerminalStatus Collaboration	17
2.8	Common UML Semantics Used in Arctis	18
2.9	ESM of the Terminal Status Activity	20
2.10	Model Transformation	21
2.11	SPACE Method and Tool Support[41]	23
2.12	Pre AJAX Web Model	25
2.13	Post AJAX Web Model	26
2.14	DOM Tree	28
2.15	An Example XML Document	28
2.16	A Servlet's Life Cycle	30
2.17	An AJAX Web Application Development with the GWT Framework	32
3.1	AJAX Integrition in Arctis	35
3.2	Interface Definition	36

3.3	Definition of the <i>Task</i> Class	37
3.4	An Implementation of the <i>MyService</i> Interface	38
3.5	Asynchronous Interface Definition	39
3.6	Generation of a Service Proxy Object	39
3.7	Remote Service Call	40
3.8	Implementation of the Call Back Methods	40
3.9	Relationship between the Interfaces and Classes of the GWT RPC mechanism	41
3.10	Working Procedure of the GWT RPC Mechanism	42
3.11	Example Server System Presenting Our Client-Server Com- munication Model	44
3.12	Role of GWT Code Generator in Arctis	45
3.13	Sample Locations of Different Parts of a Typical GWT Appli- cation Layout	47
4.1	General Scenario of a Web System	49
4.2	Use Case Diagram of the Example Scenario	50
4.3	Example UI of the Example Online Shopping System	51
4.4	Status of the Example UI While Placing an Order	52
4.5	Status of the Example UI While Confirming an Order	53
4.6	Table Summary of the Example Database	54
4.7	Entity Realtionship Diagram of the Example Database	55
5.1	Server System Model of the Example Online Shopping System	58
5.2	Animation of the Token Flow between the MySQL and the LogIn Activity Block	59
5.3	Behavior of the MySQL Building Block	62
5.4	ESM of the MySQL Building Block	63
5.5	Behavior of the LogInGUI_Proxy Building Block	63
5.6	ESM of the LogInGUI_Proxy Building Block	64
5.7	Behavior of the LogIn Building Block	65
5.8	ESM of the LogIn Building Block	66

5.9	Behavior of the ServiceQuery_Proxy Building Block	67
5.10	ESM of the ServiceQuery_Proxy building block	68
5.11	Behavior of the ServiceQuery Building Block	69
5.12	ESM of the ServiceQuery Building Block	70
5.13	Behavior of the AddtoCartGUI_Proxy Building Block	70
5.14	ESM of the AddtoCartGUI_Proxy Building Block	71
5.15	Behavior of the AddtoCart Building Block	73
5.16	ESM of the AddtoCart Building Block	74
5.17	Behavior of the OrderInfoEnabler_Proxy Building Block	75
5.18	ESM of the OrderInfoEnabler_Proxy Building Block	76
5.19	Behavior of the PlacingOrderEnabler Building Block	77
5.20	ESM of the PlacingOrderEnabler Building Block	78
5.21	Behavior of the OrderingUI_Proxy Building Block	78
5.22	ESM of the OrderingUI_Proxy Building Block	79
5.23	Behavior of the PlacingOrder Building Block	80
5.24	ESM of the PlacingOrder Building Block	82
5.25	(a)Service Specification as a Collaboration among the Participating System Components, (b)Service Specification as a Hierarchy of Building Blocks	84
6.1	Error Situation in the LogIn Activity Block	86
6.2	Simulation of the LogIn Activity Block	87
6.3	Error Situation in the PlacingOrder Activity Block	88
6.4	Two Token Flows arrive at the Join Node in One Activity Step	89
6.5	Error Situation in the Online Shopping System	90
6.6	Error Situation is Removed Placing a Timer	91
6.7	Example Server System Containing the Client-Server Communication Model	92
6.8	Automatically Generated Methods from the Input/Output Pins of the GWTGUITesting_Proxy Building Block	93
6.9	Implementation of the Client-Server Communication Model	94

6.10	Automatically Generated Methods from the Input/Output Pins of the ServiceQuery_Proxy Building Block	95
6.11	Implementation of the ServiceQuery Building Block	95
6.12	Test System Model	96
6.13	Activity and ESM of the Service1 Building Block	97
6.14	Activity and ESM of the Service1_Proxy Building Block	98
6.15	Activity and ESM of the Service2 Building Block	99
6.16	Activity and ESM of the Service2_Proxy Building Block	99
6.17	Reply from Service1 Building Block	100
6.18	Reply from Service2 Building Block	100
6.19	Current Session ID of the Client with the Firefox Browser	101
6.20	Current Session ID of the Client with the Google Crhome Browser	101

Chapter 1

Introduction

1.1 Context

After 3GL (Third Generation Programming Language), MDD (Model Driven Development) [64, 65] adds the next step of abstraction in writing software applications. In the history of computing, each higher level of abstraction has enhanced the productivity and simplified the writing of applications. MDD has enabled the creation of far more complex applications without increasing the project management workloads. It has made the building of applications faster, cheaper, and with higher quality compared with lower-level techniques [65]. MDD abstracts away the technology details from the models. So, the developer specifies the problems on a platform independent model, targets a platform and the tool automatically generates the code, thereby raising the productivity.

Let's cast an eye on the prospect of MDD in the web domain. The World Wide Web is now treated as a platform for delivering sophisticated and complex enterprise applications in several domains. However, web application developments continued to be performed in an ad hoc and technology specific ways. Underlying technologies and platforms in the web domain are changing and replacing with new ones making the problems of usability, maintainability, quality, reliability and adaptability of already developed web softwares. To solve these issues, web engineering [66] is an emerging multidisciplinary arena for systematic development of web applications. The applicability of MDD has been realized in the web engineering domain in order to address the adaptability problem of web softwares to technological changes. Following MDD approach in web engineering allows developing of web applications using separate models to describe the different concerns that constitute web

systems. Web software development covers the development of several concerns like business logic development, content management, navigational and presentation architectures development. MDD approach in the web domain allows the use of separate models to describe these different concerns. This allows performing of web applications development in a systematic format as well as adding of all the advantages to the web development those MDD provides.

In our thesis work, we have considered the model driven development of Asynchronous JavaScript and XML (AJAX) based web applications. AJAX [17, 1] is probably one of the most important hypes in the web application domain that has appeared in the last few years. With AJAX, web applications can behave like a desktop application. AJAX facilitates higher responsiveness in web applications providing asynchronous communication with server and partial updates of a web page without reloading the whole page. Gmail, Google Suggest, and Google Maps are using AJAX [17]. However, the problem with AJAX is that the developers tend to integrate the important business logic in the client side using Javascript, whereas, the web application development requires the separation of business logic from the view of a web application. GWT (Google Web Toolkit) Framework comes at this point giving a solution to this problem of AJAX based web applications. GWT enables the separation of business logic from the client side by allowing the development of client side programming in Java and creating separate packages for the client and the server. So, the server logic or business logic can now be put into the server package to run on the server system.

From the above study we have found that, in the context of model driven web application development and AJAX based web application development, we can develop the business logic in model driven way to raise the productivity of AJAX based web applications. For modeling the server or business logic we have used SPACE method. SPACE method enables the service creation from distributed systems. It models services as collaborations among participating system components. Web system is an example of distributed system, where the participating system components are the client and the server. With SPACE method, the business logic or the service provided by the server to the client can be modeled as collaborations among the client and the server. Keeping this in our mind we have started our work.

1.2 Problem Statement

Our motivation is to integrate the model driven business logic development in AJAX based web applications and at the same time to integrate the AJAX based web application development in Arctis (a tool suite of SPACE method). In order to do so, we need to perform a case study on web application domain with Arctis, where the underlying technology is AJAX and GWT framework. With GWT framework a web application uses AJAX in its client side to communicate with the server and can use Java Servlet [35] in its server side for implementing the server side logic . As a part of this case study, we will try to build up some reusable building blocks for an AJAX based web application with Arctis. We will describe a use case scenario to use those building blocks. We will try to implement our specified system and evaluate our resulting work.

1.3 Related Works

Let's have a look into what has been done so far for integrating MDD in web domain. During the last years, the web engineering community worked on methods for modeling web systems, such as, Hera [32], OOHDM [63], OO-H [25], OOWS [67], UWE [38], WebML [16], W2000 [45] and WebSA [55]. In the following sections we demonstrate the brief detail of some of these methods.

WebML [16] is a web modeling language with graphic notation and textual XML syntax produced in order to perform model driven development of complex data intensive web sites. WebML gives the facility to design the core features of a website at higher abstraction level omitting the architecture details from it. With WebML, specification of a website is devised under its four orthogonal perspectives, such as, its data content (with a structural model), the pages that compose it (with a composition model), the topology of links between pages (with a navigation model), the layout and graphic requirements for page rendering (with a presentation model), and the customization features for one-to-one content delivery (with a personalization model) [15]. Construction of website with WebML is supported with a CASE tool named WebRatio [2].

Gómez et al. developed Object- Oriented Hypermedia (OO-H) Method that provides a set of new views that extend UML to provide a Web interface model [23]. The OO-H method defines a way of migrating legacy systems

to web systems where the legacy systems' logic modules are expressed in UML class diagrams. For modeling web interface departing from the UML compliant business logic, OO-H method introduces three diagrams which are the NAD (Navigation Access Diagram), that defines the navigation and interaction view, the APD (Abstract Presentation Diagram) and the CLD (Composition Layout Diagram) that together gather the concepts related interface architecture and presentation [14]. Gómez et al. also developed a CASE environment named as VisualWADE [24] to support the methodological aspects of the OO-H method. Baresi et al. in [54] presented W2000 as a complete notation for modeling complex Web applications. W2000 acts as a homogeneous solution integrating several concepts like HDM (Hypertext Design Model, [21]), UML (Unified Modeling Language) and the model driven approach [22] in order to model all the aspects of Web applications, from Web pages to business transactions. With W2000, web application development is organized following four steps: Requirements analysis, Hypermedia design, service design and customization activities with designing four models: information, navigation, services, and presentation. Requirement analysis covers the analysis of main information and business processes requirements needed by the different users of the web application. Hypermedia design first produces the draft of information, navigation, and presentation models focusing on the essential properties of web applications which are later refined introducing all the necessary details of the web application. Running in parallel with Hypermedia design step, service design specifies the business logic of the application. Customization activities give the opportunity to specialize the web application's features like content, navigation, presentation, and services with respect to special purpose contexts. W2000 is supported by a toolset: graphical editor for designing models, MOF (MetaObject Facility) [26] repository for storing models, constraint validator for ensuring consistent model design and Rule Engine for applying model transformation rules. This toolset is implemented as an add-in to Eclipse.

UWE (UML based Web Engineering) [29, 38, 37, 9, 19, 69, 56, 57] is an approach to systematic web application development with the ultimate goal of fully automatic generation of web systems. The UWE process reflects the MDD process following MDA principals. It covers the whole development life cycle of web systems from the requirements specification to code generation. UWE is also based on several OMG standards such as UML, MOF, QVT (Query/View/Transformation) [27], OCL (Object Constraint Language) [58] for the specifications of models, description of meta models, definition of model transformations and checking the consistency of UWE models. Like other web engineering methods, UWE process separates different concerns of

the Web applications and advocates for designing separate models for each concern. The concerns addressed by UWE are the requirements, content, navigation structure, presentation, and business processes.

UWE suggests modeling the functional requirements with use cases and the business processes related to a specific use case with activity diagrams [33]. Content, navigational structures and presentation modeling is done with UML class diagrams. UWE defines a UML profile reflecting web specific notation for modeling navigational and presentation models. In the development process of UWE these functional models (models of content, navigation structure, presentation, and business processes concerns) are integrated into a big picture for the purpose of verification. In order to act as MDA, architectural aspects are integrated into this big picture resulting in an integrated PIM (Platform Independent Model) from which PSMs (Platform Specific Model) are derived. Finally, from PSMs programming codes are generated.

Tool support for developing web applications with the UWE approach is provided in openUWE model driven development environment. So far, this development environment comprises two CASE-tools - ArgoUWE [36] and UWEXML [50, 36]. ArgoUWE comes as a plug-in module of the open source ArgoUML [3] modeling tool to aid the design phase of UWE method. ArgoUWE acts as a graphical editor for modeling web applications with UWE notation. In addition it supports consistency checking of the composed models and systematic model transformation techniques of UWE method. ArgoUWE delivers the design models in XMI (XML Metadata Interchange) format. UWEXML performs semi-automatic generation of web systems through transforming the XMI formatted design models into XML documents and publishing those with the help of an XML publishing framework.

The above discussed methodologies for web systems' design are basically for modeling information system covering the relationship between the underlying contents and the user perceived views of those contents; navigation patterns reflecting the interactions with those views and presentational views reflecting the presentation of information to users. However, these methodologies are still basic in treating business processes which describe the structure and behavior of an organizational activity. Nowadays, many modern organizations such as banks, government, utilities and, generally service organizations exploit the web to provide their services. This trend requires a reciprocal relationship among the organization's business process design and the web system design. Mario et al. in [59] tried to solve this problem proposing a conceptual framework named UWA+. UWA+ extends UWA [18] framework to mitigate UWA's lack about the modeling of business pro-

cesses. UWA is a conceptual framework for modeling web system using UML based notation and W2000 methodology [51]. UWA+ introduces an integration design named π diagram which supports the coupling of the business process view from PIM (Process Information Model) methodology [60] and web system view from UWA framework. Their contribution aims to give the organizations to have the opportunity of developing complex web systems integrating the business process properly.

From our study on the methods of the systematic web system development, we can say that, many methods have been proposed so far for modeling web systems in the web engineering domain. Some methods like WebML worked on only the content navigation and presentation concerns. Methods like OO-H and UWA+ proposed the integration of two separate frameworks (business process framework and web interface framework) for covering the whole web system modeling. On the other hand, methods like W2000 and UWE themselves cover the modeling of the whole development life cycle of web systems.

In our thesis work, we have focused on modeling business process of web applications. We have tried to model the business process using an already established method and its tool suits. Our concern is only the modeling of the business process not the modeling of navigation or presentation concerns. For web interface designing or more specifically user interface designing, we are not applying any modeling framework, rather it is a framework named GWT that supports AJAX. In our thesis work, we have used SPACE method [49] for business process modeling of web applications. It has already proven its success with its tool-suits giving solutions on several domains like mobile services, embedded systems, home automation and trust management. In the perspective of the web application development, clients and servers reside on different systems. Service creation for distributed systems is one of the goals of SPACE method. So, web application is a good example for SPACE method and its tool suits (Arctis and Ramses) to study on.

1.4 Objective

Our main goal is to study the modeling of the server side business process of a web application in Arctis where the client side technology is AJAX. In order to do so, we have to do a comprehensive study on SPACE method and its tool suit Arctis. For fulfilling our main goal, our subordinate goals are pointed here

- To describe a web application scenario.
- To model some reusable building blocks specifying the server logic of the web application scenario.
- To try to implement the system model that we will design
- To evaluate our resulting work in discussion.

1.5 Contribution

In order to accomplish the goals set in the preceding section, we have first designed a model in Arctis that specifies the communication between the client and the server. Then we have presented a scenario that describes how the web application, whose business process we want to model will work. We have modeled some reusable building blocks those describe the server side business process of the presented scenario.

For the implementation of the system model that we have designed, we have proposed the functionalities that a code generator should perform during the automatic code generation from the state machines of our model. We have implemented our client server communication model to verify that our model works successfully performing the client server communication. We have implemented one of our reusable building blocks that can retrieve the available service names from an existing Online Shopping Store that sells services as products. Because of our time limitation, we have not been able to implement our whole system model. However, we have implemented a simple test system model that proves that, the implementation of our system model is possible with the implemented code generator.

1.6 Thesis Outline

Our thesis structure tries to align closely with the objective and contributions described previously. We present a coarse-grained overview of our thesis structure as following.

Chapter 2: Background

In this chapter, we have described the necessary details of all the methods and technologies those we have used in this thesis. We have described the theory of MDD and MDA. We have presented our comprehensive study on

SPACE method and its tool suits. We also have given a brief detail of AJAX technology, Java Servlet and GWT framework as underlying technologies of our work.

Chapter 3: AJAX integration in Arctis

The aim of this chapter is to show the idea that we have developed for modeling the client-server communication of AJAX based web applications in Arctis. Before showing that, we have given an idea of how AJAX based web application with GWT framework communicates to the server. For executable code generation from the service models, a code generator is required. For implementing the communication model we have developed as well as for implementing the server logic model or more specifically the server system model, a GWT code generator is needed. In this chapter, we also have proposed the functionalities those the GWT code generator should include.

Chapter 4: Scenario Description

First of all, in this chapter we have presented the common scenario that any web application holds. In this chapter, we have described an online shopping system as our AJAX based web application scenario. We have described the services, those the shopping system will provide.

Chapter 5 Development of Reusable Building Blocks

For modeling the business process or service provided by the described online shopping system, we have modeled some building blocks with reusable functionalities in Arctis. Moreover, we have described the Online Shopping System model designed with our reusable building blocks. We also have described the behavior of those building blocks in this chapter.

Chapter 6 Implementation

We have first implemented our client-server communication model. Then we have tried to implement one of our designed building blocks. Finally, we have implemented a simple test system model whose activity resembles our specified Online Shopping System model. This chapter describes our implementation attempts and shows our success results.

Chapter 7 Discussion

In this chapter, we have tried to evaluate our resulting work. This chapter also describes the limitations of our work and the future research works that can be carried out.

Chapter 2

Background

2.1 MDD (Model Driven Development)

MDD (Model Driven Development) or MDE (Model Driven Engineering) is a software development methodology that designs complex systems based on formal models from which in a series of model transformations, executable software can be generated. With the increase of the complexity of today's software systems, the usability of MDD approach is increasing among the software engineering community.

Complex software system developing in traditional software engineering approaches with 3GL requires lots of technical skills. These traditional approaches are highly technology oriented. Writing and understanding programs comprising even hundred lines of code is a difficult task let alone, million lines of code. Also, 3GLs are defect intolerant meaning that even a trivial defect can cause the most expensive failure. For example, because of a single break statement missing in a C program, the AT&T long distance network in the northeastern United States crashed in 1990 costing hundreds of millions dollars [65].

MDD specifies systems in a higher level of abstraction to avoid those above mentioned problems. This abstraction promotes simpler models with a greater focus on problem space furthering away the implementation details. This allows the domain experts only to focus on business logic. From these abstract models, then through a series of automated transformations and refinements some intermediate models with platform specific details and finally the deployable code is generated. The specifying system properties are kept consistent all along the path of model transformation and code generation. MDD

also provides the possibility of verifying and testing models enabling to correct the errors before fundamental design decisions are made [65]. Thus, raising the level of abstraction of system specification to be closer to the problem domain and raising the level of automation of code generation from models, MDD maximizes the productivity of software with quality of service.

2.2 MDA (Model Driven Architecture MDA)

MDA (Model Driven Architecture MDA) is a framework launched by OMG (Object Management Group) to provide standard guidelines in engineering software systems with MDD approach. MDA does not explicitly specify a detailed development process, rather it gives only generic method guidelines defining the system models and facilitating transformations between different model types. MDA specifies system in three layered models: system requirements are specified in the CIM (Computation-Independent Model); the Platform-Independent Model (PIM) is the model that describes the system design independent of the implementation platform; the Platform-Specific Model (PSM), on the other hand, describes the system design in the form of a platform-dependent model [8]. From PIM following an OMG standard mapping, PSMs are generated from where application code is generated. Figure 2.1 illustrates the software development lifecycle according to MDA.

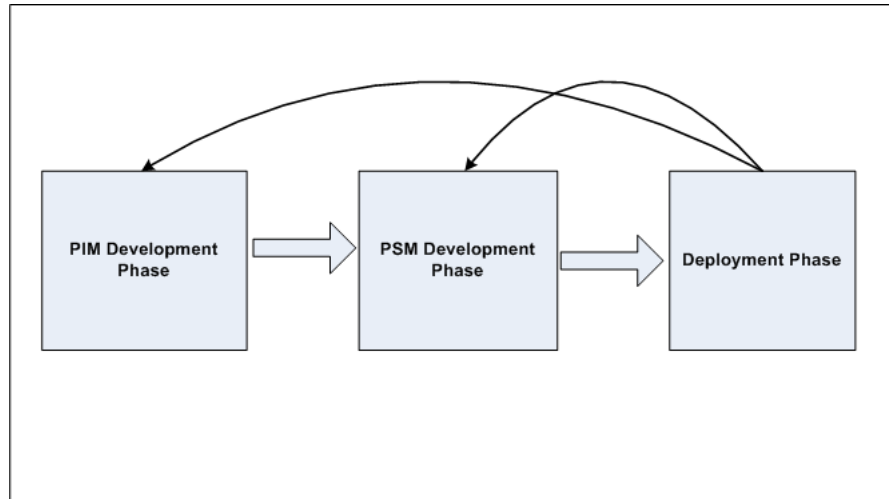


Figure 2.1: Software Development Lifecycle with MDA

MDA is actually a consolidation of several OMG standards for using models

extensively in software development. PIMs are defined using platform independent modeling languages like UML (Unified modeling language). UML Profile which is a standardized set of extensions for UML models to define PIMs with a specific domain details. For performing automated mapping between different models a model based transformation language QVT [27] is a standard. Meta Object Facility [26] is a standard for defining metamodels for other languages like UML. These and several other standards define the core infrastructure of the MDA and provide the architecture for modern system modeling.

2.3 SPACE Method

SPACE [49] is an engineering method for rapid creation of services, developed at NTNU (Norwegian University of Science and Technology) which follows MDA as a technical framework. With this method, system is specified using collaborations as specification units. The meaning of collaboration is explained in subsection 2.3.1. From that collaborative models through automated model transformations executable state machines and system components are generated which are then used as input for the code generation. These model transformation and code generation are automated processes in SPACE. Figure:2.2 illustrates the system development cycle in SPACE method.

With compositional Temporal Logic of Actions [31], SPACE provides automated model checking, verification of system specification and correctness of model transformation. The main focus of SPACE method is on composition of services using reusable building blocks. In a word, SPACE is an engineering method that facilitates the rapid creation of services specifying the services as collaborations and generating the executable code from collaborative service models through automation with necessary tool support. We describe the whole development process with SPACE method in more detail in the following subsections. As our aim in this thesis work is to develop some building blocks for a web application, we describe the service specification part more elaborately than the automated processes in SPACE method.

2.3.1 System Specification or Service Composition

According to [43], a service is an identified functionality comprising a behavior performed by participating components or entities. From this definition,

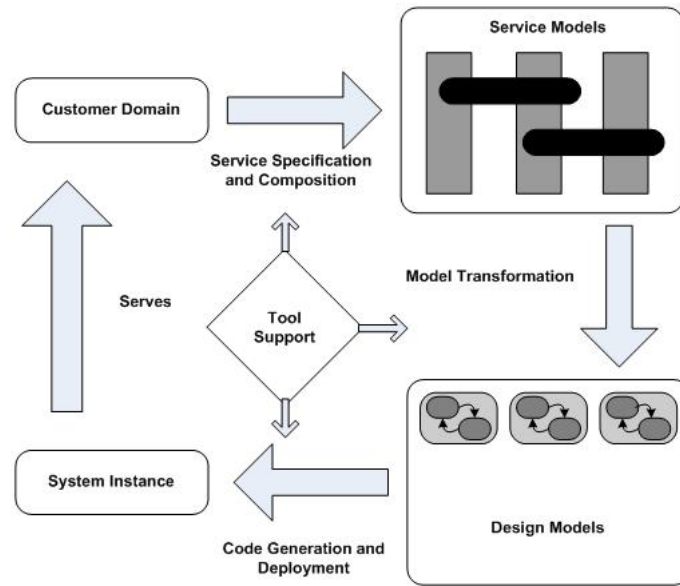


Figure 2.2: System Development Cycle in SPACE method (adopted from [43])

it is clear that, services can be specified as collaborations among several components those interact to do specific tasks. However, a service performs a specific task and requires to be combined with other services to specify the complete system. Figure 2.3 illustrates the definition of service and system.

In order to reuse a previously specified service in composing a whole system, SPACE method specifies services in a self contained way. So, services are specified as collaborations among participating components. Collaboration is actually a specification of contextual relationship among instances those interact within a context in implementing desired functionality [34]. This collaboration acting as a subservice can be used to specify another service. Thus the reusability of services is implemented.

For implementing the above mentioned conceptual theory, SPACE method specifies services using the combination of UML 2.0 collaborations [28] and UML 2.0 activities [10]. UML 2.0 collaborations describe the structure view of composed services, whereas UML 2.0 activities describe their detailed behavior.

To explain the related models at different abstraction levels in SPACE method, we took an example system specification from [43] which is a mobile alarm

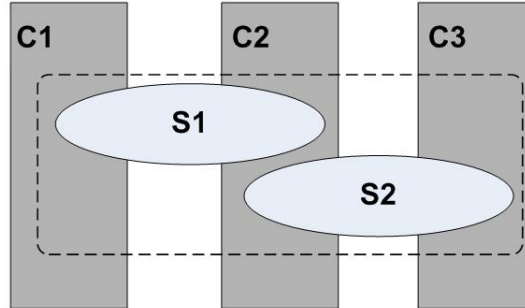


Figure 2.3: Services as Collaborations among Several Components. C depicts system components, S depicts services and dotted rectangle comprises the whole system.

system shown in Figure 2.4. When an alarm activates in a house, this system informs that to the house owner through his mobile phone. In this system, a sensor (for example, Fire detector) works to sense an abnormal situation, a camera is installed to provide the visual coverage of the sensor area and a residential gateway maintains the communication between these components and the application server of the telecom operator. To inform the house owner about the critical situation with traditional call-based features the application server of the telecom operator interacts with the call control server.

2.3.1.1 UML Collaborations Illustrating Service Structure

Figure 2.4 represents the UML collaboration of the mobile alarm system which we have described in the previous section. A collaboration specifying a system gives the structural representation of the system with collaboration roles, connectors, collaboration uses and role bindings. In Figure:2.4 the rectangular elements are collaboration roles (sensor, camera, residential gateway, application server and call control) which are the descriptions of the participating components in the example mobile alarm system. We have provided the description of these roles in this Mobile Alarm System in the previous section. The connector between two collaboration roles (for example, sensor and residential gateway) describes the relationship between them. As we said earlier, a whole system is composed of several sub services, these subservices

are referenced in the collaboration model using collaboration uses. More clearly, a collaboration use is a reference of a specific collaboration. In Figure 2.4, the elliptical curves are collaboration uses. Here, the collaboration use s1: Sensor Alarm notifies the residential gateway about the occurrence of a critical situation. t: Terminal Status finds out the availability of the house owner's mobile phone. a: Alarm dialog acting as an interactive voice service informs the situation to the house owner and asks for necessary tasks to do. s2: Send SMS collaboration use sends a SMS to house owner's mobile phone if his mobile phone is not available and s3: Security Dialog is for calling help from a security company on demand. Role bindings are dashed lines in the same figure describing the role of the participant in the system to specific subservices.

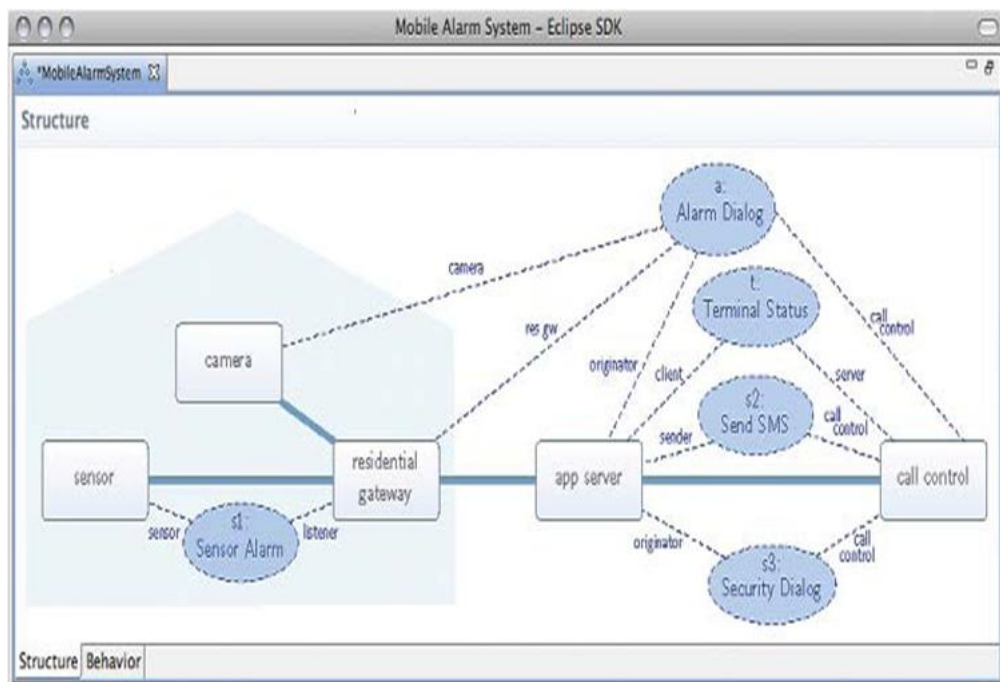


Figure 2.4: Collaboration of the Mobile Alarm System [43]

2.3.1.2 UML Activity for Illustrating Service Behavior

In SPACE method, associated with each collaboration, a behavior is attached. With UML 2.0 activities SPACE method describes this behavior which is actually a detailed local behavior of the participating components

as well as their interactions among themselves. We can think of UML 2.0 activities as a flow graph which contains logic nodes and connected edges. With this flow graph, UML 2.0 activities express the sequence, conditions, inputs and outputs to invoke or interact with other behavior or system environment [11]. UML 2.0 activities utilize token flow semantics like Petri nets [28]. Tokens traverse along the directed edges and are operated on the nodes according to the logic established by the activity. Moreover, activities express the fine grained logic of a specific task in so detailed level which is close to the implementation in a programming language [20]. An activity model is also identified as a building block.

An activity as a subordinate activity can be referred in another activity with call behavior actions. An activity containing a subordinate activity interacts with it by its input and output parameter nodes. The new activity has also additional behavioral logic and uses sub activities to describe some part of its behavior. So, this new activity forms a new level of abstraction. That is, UML activities can be used on several levels of decomposition for the specification of systems [20]. At the topmost level of abstraction UML activities focus on the overall behavior of the system with coarse business functionalities and on the lowermost level UML activities specifies the detailed behavior with fine grained logic. So, we can think of a complex system as a hierarchy of UML activities or building blocks as shown in Figure 2.5. Depending on the complexity of the system and its functionality the composition levels may increase to simplify the specification of the system.

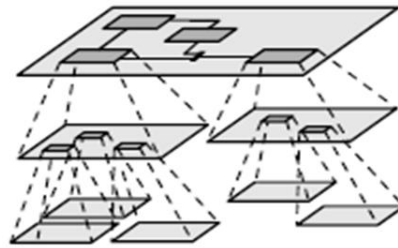


Figure 2.5: System as Hierarchy of Building Blocks

Figure 2.6 shows behavior of Mobile Alarm System collaboration shown in Figure 2.4. This activity represents the highest level of abstraction where the behavior of collaboration uses are referred with corresponding call behavior actions. Also, this activity shows the coordination of sub collaborations in a collaboration with input and output parameter nodes. We can see in the Behavior of MobileAlarmSystem Collaboration, the activity is started by a

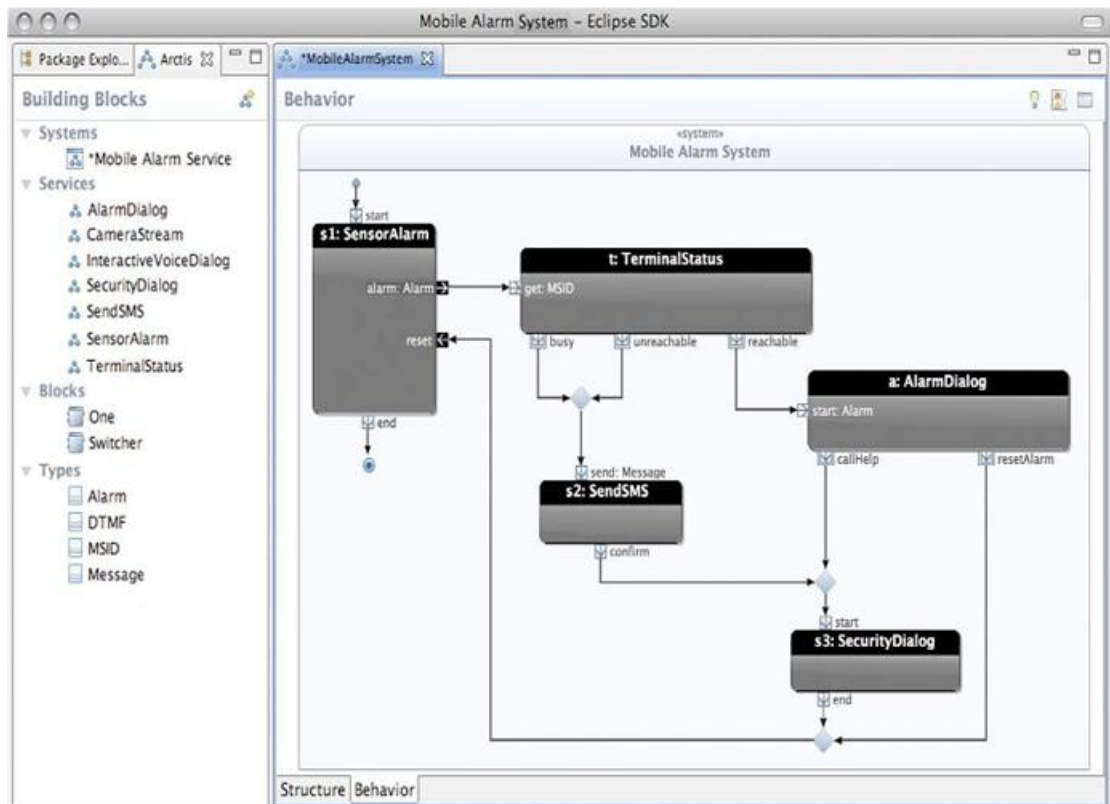


Figure 2.6: Behavior of the MobileAlarmSystem Collaboration

token arriving at the *start* node from the initial node at the upper left corner. This token activates the Sensor Alarm sub service. If an alarm occurs, the *t:Terminal Status* subservice is activated as a token is emitted via output pin *alarm* of the *s1: Sensor Alarm* sub activity. Sub activity *t:Terminal Status* finds out the availability of the house owners mobile phone giving three possible outputs. If the mobile owner's phone is reachable, *a:Alarm Dialog* sub activity is activated as token flows via pin *reachable* to *a:Alarm Dialog*. Sub activity *a:Alarm Dialog* according to phone users command can either start the reset logic of *s1:Sensor Alarm* sub activity with output pin *reset* or activate *s3:Security Dialog* with output event *call help*. If the phone is busy or unreachable the *s2: Send SMS* sub activity is started to sent SMS about the incident and the *s3: Security Dialog* sub activity is activated with output pin *confirm*. *s3: Security Dialog* can also reset the sensor alarm. Through resetting the alarm the entire activity is terminated.

Figure 2.7 shows the internal detailed behavior of the *t: Terminal Status* col-

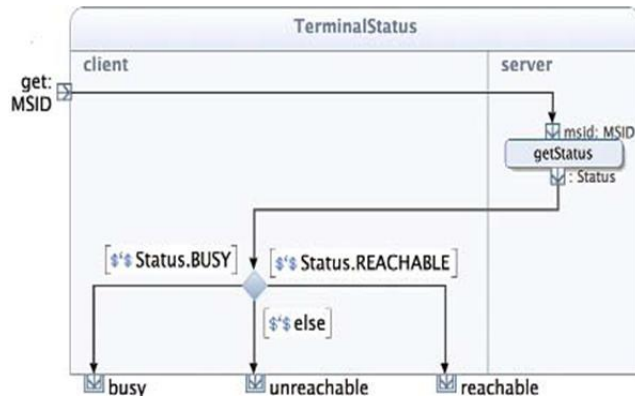


Figure 2.7: Internal Behavior of the TerminalStatus Collaboration

laboration in Figure 2.4. From Figure:2.4 we can see the t:Terminal Status collaboration is bounded to collaboration roles app server and call control by the role bindings client and server. The collaboration roles are represented as activity partitions in the internal behavior of the collaboration. In Figure 2.7 we can recognize the collaboration role client and server as activity partitions. Activity t:Terminal Status is started via input pin *get* and Mobile Subscriber ID (MSID) of client is passed to the server. Getting the MSID *getStatus* operation finds out the availability status of the mobile phone and shows result with status object. Depending on the content of status object, the token flows towards any one of the three alternative output pins *busy*, *unreachable* or *reachable* and the activity terminates.

2.3.1.3 UML Profile and Semantics of UML activities used in SPACE method

SPACE method defines a UML profile [28, 40, 39] for extending standard UML for service specifications based on collaborations and activities and component oriented specifications based on state machines. This profile also defines the constraints on UML models (collaborations, activities and state machines) used in SPACE method to ensure their semantics and executable code generation from them.

We summarize the common semantics used in activity diagram in SPACE method in Figure 2.8.

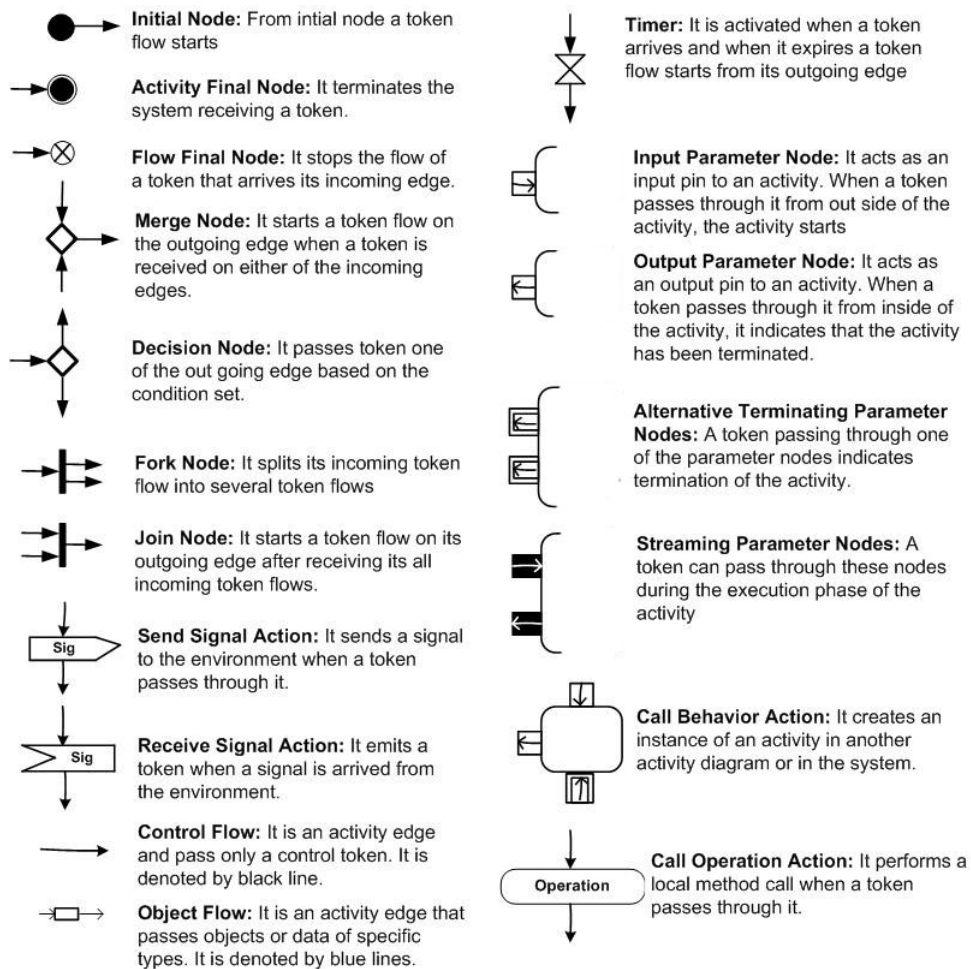


Figure 2.8: Common UML Semantics Used in Arctis

2.3.1.4 Types of UML activities Used in SPACE method

System Activity

System activity has no parameter nodes. It has at least one initial node and may have several flow final nodes. It comes with the «System» stereotype in its title. It has no ESM (External State Machine). We explain ESM in sub section 2.3.1.5.

Sub-activity

Sub-activity must have parameter nodes. It cannot have any flow final node. Unlike system activity, it must have an ESM. Sub activities are instantiated

in other activities by call behavior actions.

Shallow Building Block

It is a reusable piece of activity which has no internal behavior specified. It is specified by only its external behavior (ESM). In our thesis work, we have not used any shallow building block.

Proxy Building Block

Proxy building block or Proxy activity is a newly implemented building block in Arctis. In a local system, it represents the behavior of an outside system. This building block is specified by only its parameter nodes and ESM. It has no internal behavior details. The building blocks we have developed in this thesis work are specially based on this Proxy block. We explain the Proxy block in detail in section 3.2 in chapter 3, so that, its functionality can be understood more clearly.

In Arctis, the general term building block indicates all the above mentioned activities.

2.3.1.5 External State Machine Illustrating External Behavior of Activity

For composing systems with reusable building blocks, SPACE method encapsulates each sub activity or reusable building block with External State Machines (ESMs). ESMs are a variant of UML state machines. An ESM acts like an interface that hides the inner details of an activity and expresses the externally visible behavior of it. Encapsulation of activity with ESM allows an engineer who did not build an activity to use that activity as a reusable building without looking into the inner details of it. An ESM describes the order in which tokens can pass the various parameter nodes of an activity [20]. This information is necessary to correctly build a system from reusable building blocks. Figure 2.9 shows the ESM of the t:Terminal Status activity shown in Figure 2.7

The labels on the arrows correspond to parameter nodes of the activity. A token passing these parameter nodes causes transitions of the ESM. A separator "/" distinguishes the parameter nodes as triggers or effects. Triggers are invoked by a token from the outside of the building block, whereas effects are invoked by a token by the internals of the building block. From Figure 2.7 we can see that, MSID arriving at pin *get* from outside of the activity activates the activity. So, In Figure 2.9 the transition is specified as *get/* indicating pin *get* as a trigger. Then the activity goes into the retrieving

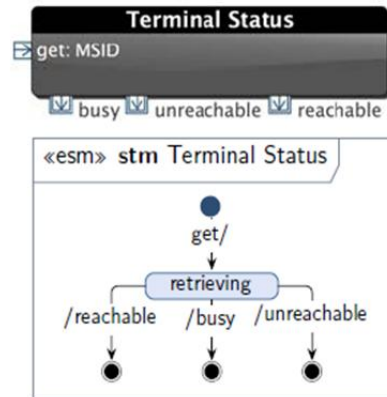


Figure 2.9: ESM of the Terminal Status Activity

state for finding out the availability of the phone. From the retrieving phase the termination of the activity is happened when a token passes one of the three pins *reachable*, *busy* and *unreachable*. This termination transitions are labeled as */reachable*, */busy* and */unreachable* in the ESM indicating *reachable*, *busy* and *unreachable* pins as effects.

The services specified as collaborations and encapsulated as UML 2.0 activities and ESM interface descriptions can be stored in the library of different domains. Later, these services can be used as reusable building blocks from the domain specific libraries for the system specification in a particular application domain. In order to do so, the required service or more specifically, subservice is picked up from the library and instantiated and enclosed in a collaboration to compose a new collaboration or system specification.

2.3.2 Formal Analysis and Automated model Checking

To verify the behavioral properties of the service specifications, the semantics of UML collaborations and activities are formalized automatically in the form of cTLA [31, 44] specifically cTLA/c, which is based on the linear-time temporal logic TLA [52]. This formalization helps the building blocks and their compositions to be analyzed thoroughly with an automatic model checker. With automatic model checking, all possible states in a specification are searched and analyzed. This analysis is based on properties that must be fulfilled by any service specification to be consistent [43]. This formal analysis is hidden from the developer who is working on service specification.

If any property is violated error trace of states and transitions that lead to the violating state is reported in the form of states and steps of the UML activity with a tool support.

2.3.3 Model Transformation

After specifying the system or services with collaborative specifications in combination with UML 2.0 activities and describing their external behavior with ESMs, the subsequent implementation steps are automatic.

After completing the service specification, the activities are transformed automatically into intermediate models, which are UML 2.0 state machines of participating components. Each activity partition in activity model corresponds to a participating component in the service specification. So, each activity partition expresses a part of collaborations which are bounded to that activity partition or component. In the automatic model transformation step, each activity partition is transformed into a separate state machine representing the behavior of the corresponding component implied by the behavior in that activity partition [43, 45]. Figure 2.10 gives a visualization of the model transformation. These intermediate state machines for executable components are fed into the next step as input for the code generation of execution platforms. Similar to activity models, the behavior of state machines are formalized using *cTLA/e* [46]. With this formalism, in SPACE method, it is ensured that the transformation from UML activities to state machines is correctness-preserving.

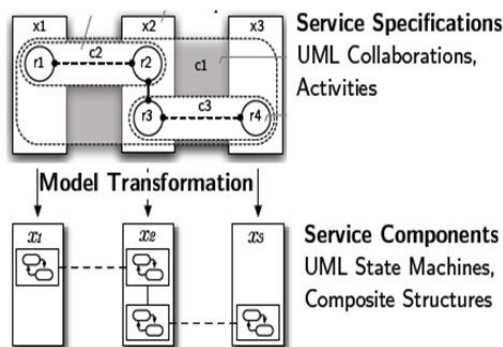


Figure 2.10: Model Transformation

2.3.4 Code Generation

The state machines can be directly used as input for generating executable implementations for several platforms. These state machines have such a form that can be easily implemented with a run time support system [13]. In SPACE method, executable code can be generated from state machines automatically with appropriate tool support. We describe the tool-suits of SPACE method in the following section.

2.4 Arctis and Ramses Tool-Suits to Support SPACE method

Arctis and Ramses are tool-suits to help the developers developing and implementing application systems or services using SPACE method. These tool-suits are implemented as a set of eclipse plug-ins. Figure 2.11 illustrates these tool-suits and their functionality to support the system development with SPACE method. Arctis aids with service specifications and compositions using UML 2.0 collaborations, activities and ESMs; model transformations from UML activities to executable state machines; and formal analysis for verifying the consistency of service specifications and model transformations. On the other hand, Ramses supports code generators for the implementation of executable state machines on several platforms.

The library of building blocks of different application domains and the editor for specifying service from scratch or using the building blocks from libraries are the components of Arctis exposed to the developers to interact with. UML standard has not its own action language, so here, Java is integrated with UML to express operations within UML activities. In order to do so, a java class is associated with each building block or each activity partition of a building block. Also, each UML operation in the activity corresponds to a java method of that dedicated java class of the activity partition. For this Java and UML integration, features like automatic code completion and correction of Eclipse Java Development Tools get available in Arctis.

There is an inspector framework integrated in Arctis providing the facility of adding numerous inspectors as plug-ins to Arctis. These inspectors perform lightweight model checking like syntactic error checking. Inspectors do this error checking directly working on the UML model. These inspectors also can check the validity of the models against the constraints of UML profile, thereby aiding the developers with application domain specific constraints

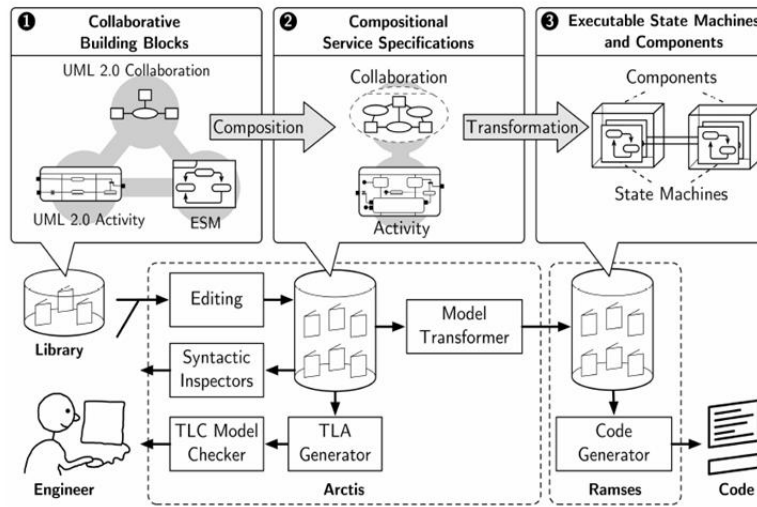


Figure 2.11: SPACE Method and Tool Support[41]

checking. Developers are notified with the diagnosis result of inspectors as error messages and optional solutions.

To make sure whether the specifications are consistent or have their intended meaning, so that, they can be converted into executable components, semantic analysis is required. Semantic analysis can be done by doing model checking on the state spaces of the specifications. As the model checkers need formal specifications of the models as their input, Arctis has a formulator named TLA Generator to do so. It has been mentioned earlier that in SPACE method activity models are formalized with cTLA formula for doing semantic analysis or more specifically formal analysis. The TLA generator performs this formalism and automatically transforms an activity into TLA+ [53] which is a temporal logic language. The properties those should be hold by the specifications are also formalized with this TLA generator. Once a specification is free from syntactic errors with the help of inspectors, the activity models are formalized with the help of TLA generator. Then TLC [68], a model checker takes the formal specifications of the activity model and the properties and performs thorough analysis on all possible state space of the specification. If any violation of property occurs the trace of the state towards the violation of theorem is projected to the Arctis editor. Arctis also supports the simulation of activities with animations, so that, the user can analyze the error situations.

For running the service or system on the service execution platform, the be-

havior description of each and every individual participating component is essential. We have described earlier that, SPACE method gives the way of behavioral description of each participating components in a system through model transformation. In Arctis, model transformation module creates a state machine by reachability analysis on the states modeled by a single activity partition [18]. Thus, from each activity partition in the activity model the behavior is mapped to each corresponding UML 2.0 state machine with Arctis's model transformer. The model transformer performs this transformation as refinement step from activities to state machines, so that, the property and behavior of activities are preserved correctly in state machines. Formalization of state machines in cTLA/e descriptions gives the way to perform these correctness-preserving refinement steps.

In order to implement the state machines on execution platforms, Ramses contributes code generators to create executable systems [13]. Everyone can develop a code generator for a execution platform and integrate this with Ramses defining extension points in Eclipse for that code generator. Code generators of specific execution platforms take UML state machines of system specifications as input and generate codes to be executed. In our thesis work, the execution platform is AJAX with GWT framework. We need a GWT code generator to generate the code from the state machines of our service specifications and to implement an AJAX web application with GWT framework.

In order to develop an AJAX web application with Arctis, we have studied AJAX. We also have studied JavaServlet and GWT Framework, as we use those technologies in the server side and the client side respectively. The detail of AJAX, JavaServlet and GWT technology are described in the following sections.

2.5 AJAX (Asynchronous JavaScript and XML)

In 1990's user interaction in web applications was request-wait-response based, which slowed down the user interaction considerably. The concept of AJAX was first coined by Jesse James Garrett in 2005 in his article "Ajax: A New Approach to Web Applications" [17, 1]. AJAX came as a boon to the web with providing desktop application like user experience. We describe in the following sub sections the magic behind AJAX and its basic technical details.

2.5.1 Working Principle of AJAX

The classic web application model, addressed as pre AJAX web model is not good for considering web as software applications. In a pre AJAX web model, user interaction triggers a HTTP request to the web server. The server performs necessary processing for example, retrieving data or doing some calculations etc. When the processing is completed the server returns an HTML page to the client. The problem is that, during the server processing time, the user has nothing to do, but waiting for a page to be loaded or refreshed from the server.

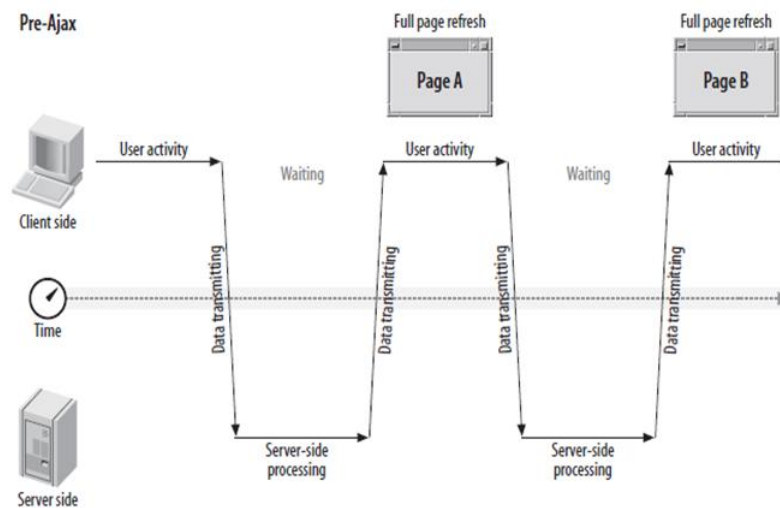


Figure 2.12: Pre AJAX Web Model

AJAX increases the web page's interactivity, speed, and usability in order to provide richer user experience. AJAX places an AJAX engine between the client and server. This engine is written in JavaScript and behaves like a hidden frame. AJAX engine renders the user interface and handles the communication between client and server. The client-server communication with AJAX is asynchronous. Asynchronous communication means the client does not need to wait for the server response. After sending the request to the server the execution in the client program does not halt, rather the execution is continued. The response is notified to the client when it is available. The AJAX engine sends requests to the server on behalf of the client and receives data or responses from the server. In a web model with AJAX, server sends small data instead of the HTML page. AJAX engine shows that data or response by updating the page partially. Thus user is free

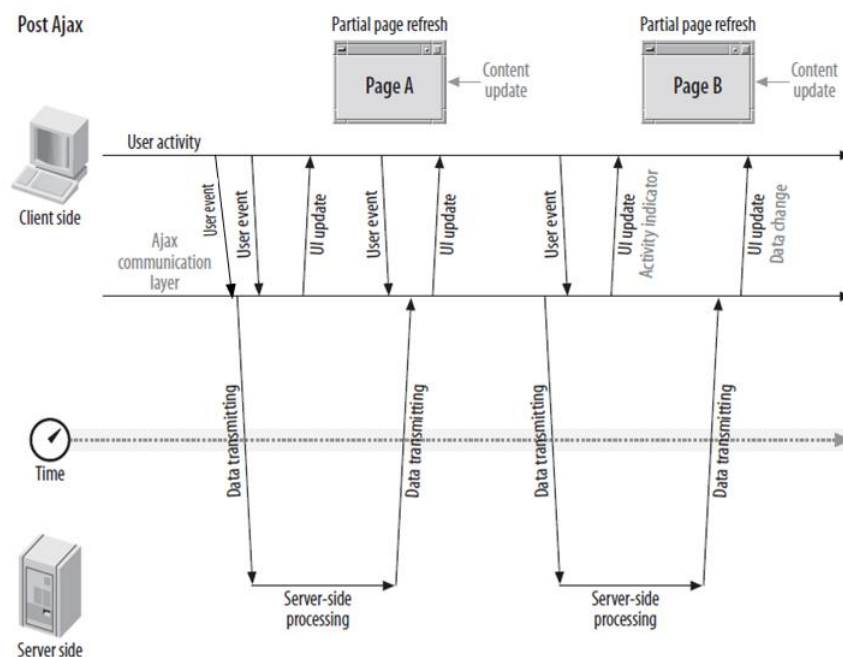


Figure 2.13: Post AJAX Web Model

to do other interaction after sending a request to the server. Figure 2.12 and 2.13 show the model of Pre AJAX and Post AJAX web models to clarify the above described idea.

2.5.2 What actually AJAX is

Ajax is not a technology in itself, but rather an umbrella term used to describe how several existing technologies such as JavaScript, the Document Object Model (DOM), and Extensible Markup Language (XML) can be used together to create Web applications that are more interactive and that remove the need for entire web pages to be refreshed when only part of the page is changing [17, 1].

AJAX incorporates the following techniques to enhance the user interaction on web.

- XHTML (eXtensible HyperText Markup Language) and CSS (Cascading Style Sheet) for providing standard-based presentation.

- DOM (Document Object Model) for facilitating dynamic display and interaction.
- XML (Extensible Markup Language) and XSLT (eXtensible Style Sheet Language Transformations) for interchanging and manipulating data.
- XMLHttpRequest for retrieving data asynchronously.
- JavaScript for binding everything together.

In the following subsections we describe a brief overview of these standards and their usability in AJAX.

2.5.2.1 XHTML and CSS

XHTML is HTML's (Hyper Text Markup Language) successor. It is a cleaner and stricter version of HTML. XHTML is actually HTML defined as an XML document. With XHTML standard HTML page should follow the strict syntax of XML. This ensures the correctness of an HTML page, whereas, with HTML there is the possibility of missing enclosing tags, breaking nested sequence like bad HTML. For AJAX, XHTML is a requirement for the presentation of the web page, as if the page is not correctly formed, AJAX cannot access different part of the web page to perform partial updates.

CSS are the templates behind HTML pages that describe the presentation and layout of the text and data contained within an HTML page [17, 1]. For partial presentational update of page CSS are useful, as its properties can be accessible via DOM to reflect the presentational change on the web page dynamically.

2.5.2.2 DOM

DOM represents the web page as a hierarchy or tree structure for accessing and manipulating HTML documents. Figure 2.14 shows the tree structure of an HTML page. DOM is a standard way for all browsers to represent the page. With DOM, it is now possible to make changes on the existing elements in HTML page or add new elements to it dynamically using scripting languages (JavaScript, VBScript), as a result, the browser can update the page or sections of it instantly. Ajax depends on these capabilities greatly to provide the rich user experience.

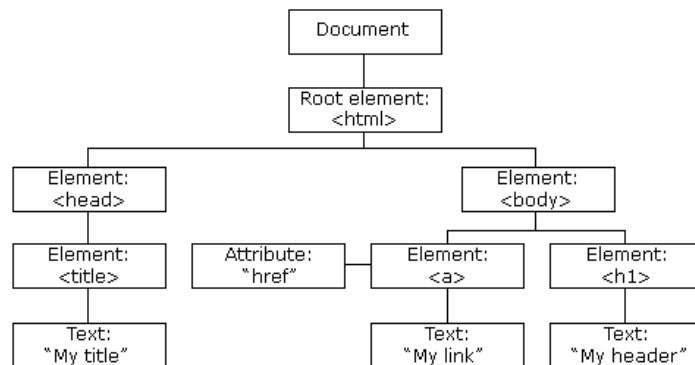


Figure 2.14: DOM Tree

2.5.2.3 XML and XSLT

XML is a markup language much like HTML, but XML was designed to structure data. The focus of HTML is on displaying the data, however, the focus of XML is on the data itself. XML was created to structure, store, and transport information. Figure 2.15 shows an example of XML which is a message sent from Tove to Jani. As XML data is stored in plain text format, this provides a software- and hardware-independent way of storing and sharing data. In AJAX, usually servers transfer data in XML format. However, XML is not the only way to transfer data from server in AJAX, text can also be transferred.

```

<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
  
```

Figure 2.15: An Example XML Document

XSLT is a language for transforming XML documents into other XML or

HTML documents or pure text. For conducting the transformation XSLT document contains a set of rules. XSLT uses another language, XPath, to query the XML document when applying its transformations [17, 1]. XSLT and XPath then help to display the section of data dynamically from XML using the browser.

2.5.2.4 XMLHttpRequest Object

XMLHttpRequest object provides the asynchronous part of AJAX. It allows requesting and receiving new data from a server in the background without any page reloading. It provides the functionality to send request for a data and notify later when the data is available. XMLHttpRequest Object does the magic part of AJAX facilitating a higher responsiveness of web pages than ever before.

2.5.2.5 JavaScript

JavaScript is a client side scripting language. It allows the creation of dynamic web pages providing a new level of interactivity. However, JavaScript is not useful to request for everything that it might eventually need from the server, because this would cause a very long loading time. XMLHttpRequest and DOM are used in JavaScript to provide the asynchronous and partial page update functionalities which AJAX offers. So, JavaScript is an essential piece of the Ajax package gluing the other technologies of AJAX together and making them functional for providing a rich user interactivity.

2.6 Java Servlet

A Servlet is a Java class which conforms to the Java Servlet API and runs in a server application (Apache Tomcat, Jetty etc) to answer client requests [35]. The Java servlet API provides a framework for building applications on web servers. Java servlet interacts with the application server. When a request comes to a servlet, the application server loads the servlet class and runs the servlet. Then, it calls the service method (implementation of application logic) of the servlet in a new thread. So, the application server handles each client request to a Servlet creating a new thread. The application server also implements the interfaces provided by the Java Servlet API to handle the session tracking and state management.

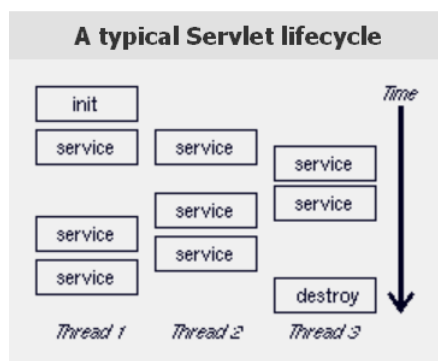


Figure 2.16: A Servlet's Life Cycle

Figure 2.16 shows a Servlet's life cycle. The `init()` method for initializing the Servlet is called only once in the whole life cycle of the Servlet. Its `service()` method is called for every request to the Servlet. The method is called concurrently i.e. multiple threads may call this method at the same time. When the Servlet needs to be unloaded the `destroy()` method is called. This method is also called only once during the Servlet's lifecycle.

2.7 Google Web Toolkit (GWT)

In May 2006, Google released GWT, an open source framework for web developers to develop Ajax application (client side code) in java instead of JavaScript.

Today's RIAs (Rich Internet Applications, i.e. desktop like web applications) are getting complex and large in size. Writing AJAX applications in JavaScript is error prone. Managing large applications in JavaScript is complex, difficult and directs to an entirely new discipline. Also, JavaScript behaves differently on different browsers. Developers spend a lot of valuable time having to code for browser differences instead of focusing on real application logic. Also, Developers tend to mix the business logic in the view of web applications using Javascript.

RIA application development with GWT gives Java developers to reuse their existing expertise and best practices. GWT gives ease of developing large applications, as Java was designed to make large applications manageable in object oriented fashion. With GWT, besides, having all the advantages of Java as a programming language, developers can use a gazillion of Java de-

velopment tools that already exist. They can use their favorite IDE, perform compile time checking, unit testing and even continuous integration. GWT also handles all browser-specific quirks meaning that compiled GWT application runs inside any modern browser (assuming JavaScript is turned on), so that developers can focus on the application logic [12]. GWT basically translates all the Java UI code to JavaScript. However, it does not mean that the old JavaScript code or application will become useless. GWT still allows interacting with existing JavaScript code as well as integrating with existing server side services. Another important functionality GWT provides is that, it separates server side logic from client side creating separate packages for the client and the server.

Google provides a plugin for Eclipse which handles most GWT related tasks in the IDE including creating projects, invoking the GWT compiler, creating GWT launch configurations, validations, syntax highlighting, etc [12]. In our thesis work, we have used a GWT SDK of version 2.0 plugin to work in Eclipse environment.

2.7.1 GWT Components

GWT provides a comprehensive set of tools including UI components to configuration tools to server communication techniques and this help web applications look, act, and feel like full-featured desktop applications. Major GWT components are as follows.

- **GWT Java-to-JavaScript Compiler**
This is the core part of GWT. This compiler converts Java code into JavaScript code in such a way, that the compiled JavaScript can run on the major internet browsers. The supported browsers include Internet Explorer, Firefox, Mozilla, Opera, and Safari.
- **JRE emulation library**
To provide developer to use some classes of core Java, GWT includes JRE (Java Runtime Environment) emulation library. This library supports some classes from *java.lang* and *java.util* packages.
- **GWT Web UI class library**
GWT ships with a large set of custom interfaces and classes for creating widgets and panels. Widget is some sort of control used by a user, and a panel is a container into which controls can be placed [12].

Figure 2.17 illustrates an AJAX web application development with GWT framework.

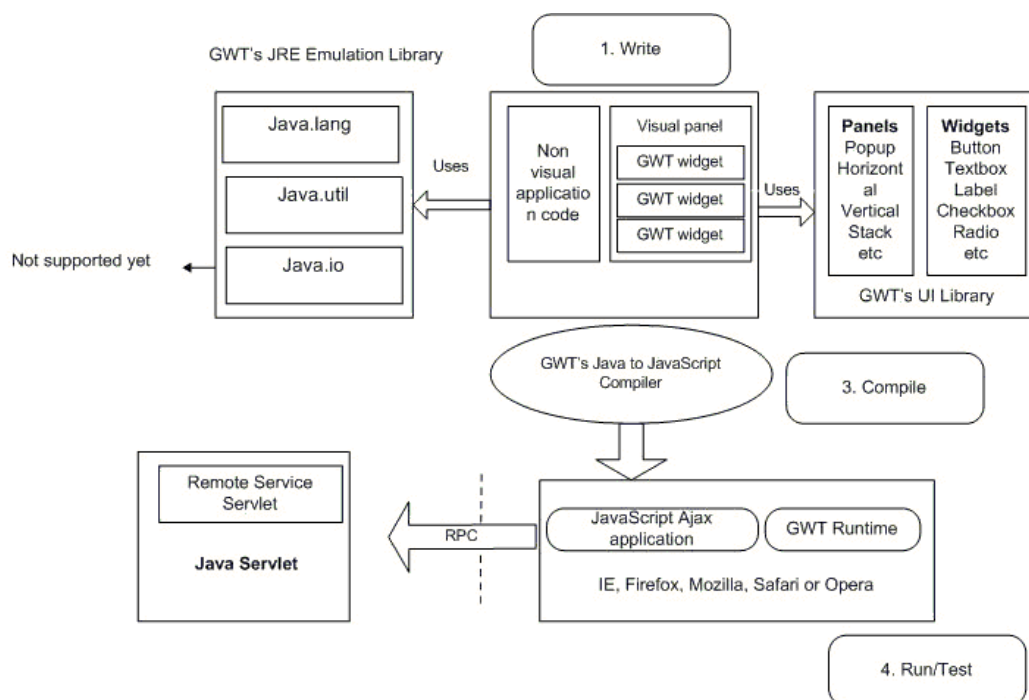


Figure 2.17: An AJAX Web Application Development with the GWT Framework

2.7.2 GWT Modes of Running

GWT applications can run on two following modes.

- **Development Mode (Hosted Mode)**
This mode can be addressed as debug mode also. With this mode, GWT allows developers to debug their application as in any Java application. In this mode, Java code is executed and widgets are displayed in a host window that emulates a web browser [12].
- **Web Mode**
This mode executes the JavaScript code generated from the compilation of client side Java code. The Web mode is actually the deployment of Ajax web application with GWT framework on a genuine application server.

In the following chapter we describe elaborately our idea on developing AJAX web applications in Arctis. In the next chapter, we also describe our related study that has been assisted us for developing our idea.

Chapter 3

AJAX Integration in Arctis

We have wanted that we will develop an AJAX web application with the GWT framework in Arctis in such a way that the developer will have the flexibility to design the user interface using the GWT UI (User Interface) elements. GWT provides a rich set of UI elements and we have wanted to utilize that in our developed web application. Currently, it is not possible to add the web UI element directly in Arctis. So, for client side modeling, we had thought of integration of a GWT UI element designer into Arctis. However, it is a time consuming research which was not possible within our assigned time limitation. Then we have decided only to model the server side logic of an AJAX web application in Arctis. As in GWT, the client side code and the server side code reside in separate packages, it is possible to put the generated code from our model into the server package by a code generator, and then execute and implement the web application accordingly. Figure 3.1 shows the proposed way of integrating the modeling with Arctis in the development of an AJAX web application.

So, according to our idea, we have developed the server side logic in the model driven way using Arctis tool, whereas we have done the client side programming manually.

In order to model the server logic in Arctis, we have found the most difficulty point is to model the communication between the client and the server. In this regard, we have studied the client-server communication mechanism in an AJAX web application with GWT framework. We describe our study briefly in the following sub sections. We also think that, it is important to know the GWT client-server communication mechanism to understand the communication model that we have designed.

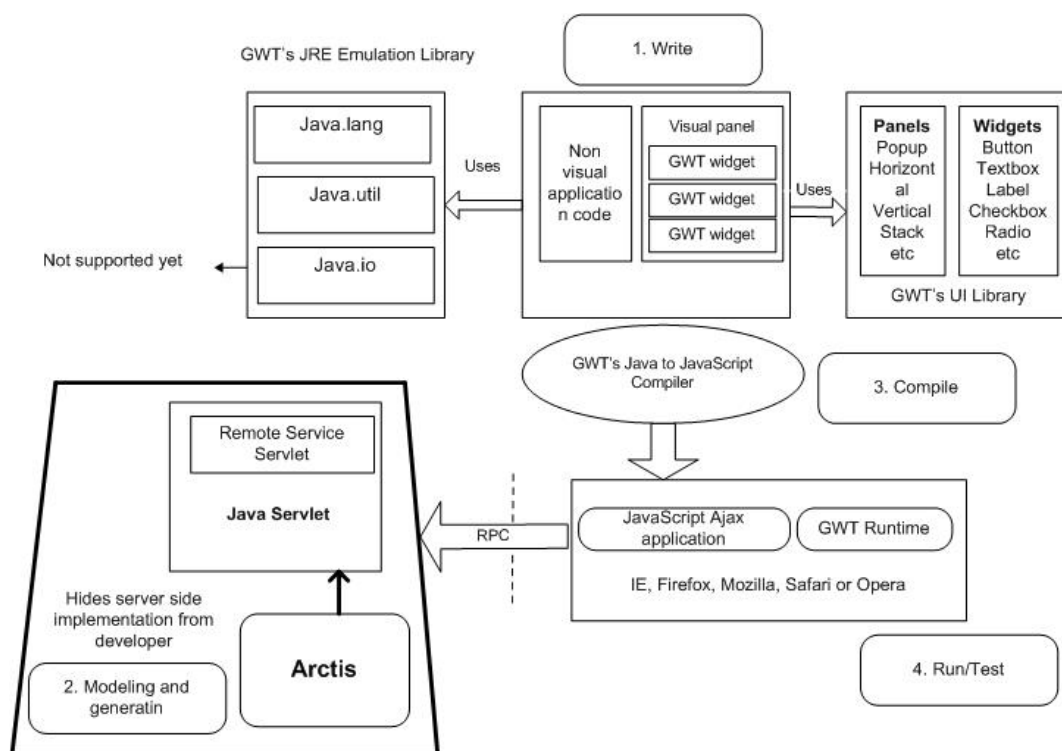


Figure 3.1: AJAX Integritin in Arctis

3.1 Client-Server Communication with GWT Framework

In section 2.5.2.4, we have mentioned that, the XMLHttpRequest JavaScript object allows an asynchronous communication between the browser client and server without forcing page refresh. GWT provides two tools those sitting on top of this XMLHttpRequest object facilitate asynchronous communication between the browser client and the server. One is Basic AJAX [12] and another is GWT RPC (Remote Procedure Call) [12, 61]. Though both mechanisms provide abstractions over protocol details of communication of a message between the client and the server, GWT RPC provides more abstraction between the two. GWT RPC allows the developer to communicate with the server by only calling methods from the client side and invoking those methods on the server side. In our thesis, our AJAX web application that we have developed is based on RPC (Remote Procedure Call) or more specifically, GWT RPC for the client-server communication. So, in the fol-

In the following section we present a brief detail of the working functionality of the GWT RPC .

3.1.1 GWT RPC

RPC is the mechanism that allows a program to execute a program on another computer and return the results of the calculation [61]. With GWT RPC, this task is accomplished by providing the developer with an interface of methods that can be called on the server similarly to regular method calls. For enabling the client-server communication in an AJAX web application with GWT framework requires the following vital 4 steps to be performed.

- Defining the Interface
- Implementing the Server Side
- Defining the Asynchronous Interface
- Deferred Binding and Remote Service Calling

We describe a brief overview of these steps in the following sub sections.

3.1.1.1 Defining the Interface

In the client side, a Java interface is defined to formulate the contract of the communication between client and server. The Java interface actually defines the signature of the methods that must be implemented in the server side. With this technique, the client and the server both know which method can be called and what is expected in return. Defining the interface is very straightforward and shown in Figure 3.2 .

```
public interface MyService extends RemoteService {  
    List<Task> getTasks();  
}
```

Figure 3.2: Interface Definition

MyService is an interface that defines the method *getTasks()*, which the client application need to call remotely. To make GWT RPC communication possible, some requirements are needed to be fulfilled. First of all, we should place the interface inside the client package of the GWT application in order to make it available for the client side application. Secondly, it should extend the interface `com.google.gwt.user.client.rpc.RemoteService` that comes with the GWT RPC library. This will make the GWT compiler understood that an RPC interface has been defined. Finally, the methods in the interface can only use classes as arguments or return types that GWT knows how to send over a wire [61]. That is, the classes should be serializable. In Figure 3.2 the *getTasks()* method's return type is a list of *Task* objects. *Task* is a user defined class and it implements a `Serializable` interface [4] to conform the GWT RPC requirement. In Figure 3.3 the definition of the *Task* class is given.

```
public class Task implements Serializable {  
  
    private Long id;  
    private String title;  
    private String description;  
    /* getters and setters */  
    ...  
}
```

Figure 3.3: Definition of the *Task* Class

3.1.1.2 Implementing the Server Side

GWT RPC implementation connects to a Java Servlet. So, the service interface defined in the client side needs to be implemented in a Servlet in the server side. GWT RPC requires the Servlet to extend `com.google.gwt.user.server.rpc.RemoteServiceServlet` class provided by the GWT RPC library. Extending *RemoteServiceServlet* by the Servlet allows the GWT RPC to abstract away the request receiving and response processing details. It even determines the method that needs to be invoked and invokes it [12]. So, the only task left to the developer is to implement the methods of the service interface in the servlet. Figure 3.4 shows a simple implementation of our

MyService interface. The server-side code that gets invoked from the client

```
public class MyServiceImpl extends RemoteServiceServlet implements MyService {
    public List<Task> getTasks() {
        /* ... retrieve a list of tasks
        List<Task> tasks = ...
        return tasks;
    }
}
```

Figure 3.4: An Implementation of the *MyService* Interface

is often referred to as a service. So, in the rest of our writing, we will address the implemented methods in the Servlet as services.

3.1.1.3 Defining the Asynchronous Interface

To use the GWT RPC asynchronous communication mechanism the client application needs to define an asynchronous version of the service interface, which should also be included in the client package. As in asynchronous communication the client side execution will not be blocked after calling the remote service, the asynchronous interface places a callback hook that gets executed when the called method running in the server returns a value. According to the GWT RPC requirement the asynchronous interface will have the same name as the original service interface name with "Async" appended to it. The methods in the interface must match all of the method names in the original service interface. However, the signature of those methods is needed to be changed. The return type of each method will be void and an extra parameter `com.google.gwt.user.client.rpc.AsyncCallback` will be added to the method. Figure 3.5 shows the definition of the asynchronous interface.

3.1.1.4 Deferred Binding and Remote Service Calling

The GWT RPC abstracted away the remote service calling (calling services running on servers) by the local method calling in the client side. However,

```
public interface MyServiceAsync {  
    void getTasks(AsyncCallback asyncCallback);  
}
```

Figure 3.5: Asynchronous Interface Definition

the remote service calling is performed by a service proxy object whose class definition is generated automatically by the GWT compiler. This service proxy object actually performs the required processes of converting the local method call into a remote method call. Figure 3.6 shows the generation of a service proxy object. The general explanation of the code snippet in Fig-

```
MyServiceAsync service = (MyServiceAsync) GWT.create(MyService.class);  
((ServiceDefTarget) service).setServiceEntryPoint(GWT.getModuleBaseURL() + "tasks");
```

Figure 3.6: Generation of a Service Proxy Object

ure 3.6 is that, the proxy object *service* is generated having the *MyService* interface definition and casted to the *MyServiceAsync* interface to be able to call the asynchronous versions of *MyService* interface's methods. The proxy object is casted to the *ServiceDefTarget* interface (resides in the GWT library) for setting up the Servlet URL to perform the remote service call. Now, the turn of calling the remote service comes. It is done by just calling the asynchronous versions of the original methods by the proxy object. Figure 3.7 shows how the *getTasks()* service is obtained in the client side. The explanation of the code snippet in Figure 3.7 is that, the proxy object *service* created in Figure 3.6 calls the asynchronous *getTasks()* method defined in the *MyServiceAsync* interface in Figure 3.5. The *TasksCallback* instance in Figure 3.7 implements an *AsyncCallback* interface to handle the result of the asynchronous method call. Figure 3.8 shows an example of such implementation. The *AsyncCallback* interface has two methods that must

```
service.getTasks(new TasksCallback());
```

Figure 3.7: Remote Service Call

```
public class TasksCallback implements AsyncCallback {  
    public void onSuccess(Object o) {  
        //Do something on the returned value o from the server  
    }  
    public void onFailure(Throwable t) {  
        //Give some notification  
    }  
}
```

Figure 3.8: Implementation of the Call Back Methods

be implemented which are *onSuccess()* and *onFailure()*. If the call is successful, then the *onSuccess()* method is called, and it receives the return value of the remote method call. If some error occurs the *onFailure()* method is called. Developers actually implement these two methods according to the application requirement to do some processing with the returned value from the server. Figure 3.9 illustrates the relationships between the interfaces and classes involved in the GWT RPC mechanism. Figure 3.9 shows that on the client-side, the generated proxy object implements both the *MyServiceAsync* interface and the GWT's *ServiceDefTarget* interface. *MyService* interface extends the GWT's *RemoteService* interface. On the server-side, the *MyServiceImpl* instances implement the *MyService* interface and extend GWT's *RemoteServiceServlet*.

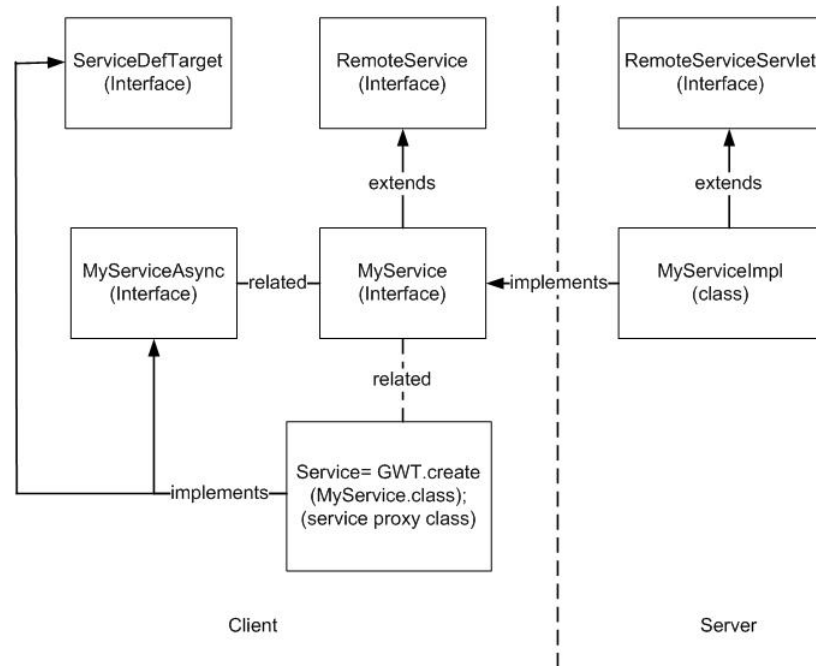


Figure 3.9: Relationship between the Interfaces and Classes of the GWT RPC mechanism

3.1.2 How GWT RPC Works

The GWT-RPC implementation works by automatically providing a proxy object for a server interface [62]. We have discussed in the previous section how this proxy object is generated in the client application. The client application uses the proxy object to communicate with the server by calling methods on the server. In section 3.1.1.4 we have illustrated how the remote calling is done. The server handles calls from the proxy and dispatches them to the corresponding Java method implementations. Return values are sent from the server back to the client's proxy. We have shown in section 3.1.1.4 that, the client application provides the proxy with a callback object. This callback object receives the return value or the failure message from the server. Figure 3.10 illustrates how the GWT RPC works.

In Figure 3.10 the *MyServiceClientImpl* is a client application instance that makes RPC method calls to the server and handles the results in a callback method. That is, *MyServiceClientImpl* performs the tasks of deferred binding and remote calling as explained in section 3.1.1.4. The Service proxy (explained in section 3.1.1.4) processes the call and send it to the server. On

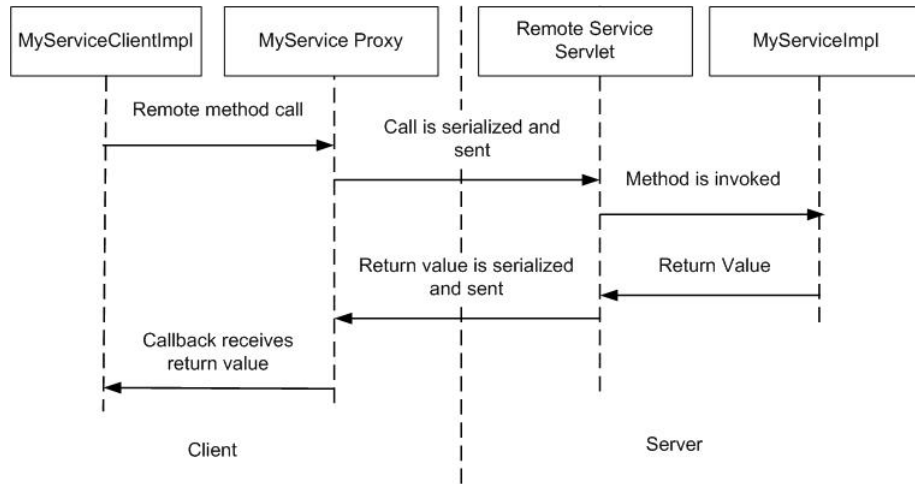


Figure 3.10: Working Procedure of the GWT RPC Mechanism

the server, an instance of the GWT's *RemoteServiceServlet* class handles the request sent from the service proxy and invokes the requested method on the *MyServiceImpl* instance.

Having understood the client-server communication procedure using the GWT RPC, we have developed the idea of modeling the client-server communication of the server side model that we want to design in Arctis for an AJAX web application. In the following sub section we explore our developed idea.

3.2 Development of the Client-Server Communication Model in Arctis

In the client-server communication process the participating system components are the client and the server. As we only have considered the modeling of server side logic, the local system on which we are working on represents the server. In our communication model the Proxy building block of Arctis represents the client behavior. We elaborate the reason behind of representing the Proxy building block as a client in the next text.

The Proxy building block is a local building block in Arctis. It gives the facility to a system to interact with the system outside of it. This block works as a proxy of the outside system. For developing a system with the proxy building block, we just need to know what we want to send to the proxy

of the outside system and what we want to receive from the outside system through the proxy. This sending and receiving is modeled by adding a input and a output pin to the proxy. Another thing we need to do is of building the external state machine to define its external behavior. The magic point is that, we do not need to know the internal behavior of the Proxy building block.

We have used the Proxy building block in our work to model the client-server communication. From the study of client-server communication in the previous sub sections, we have found that, in AJAX web applications which are based on the RPC communication mechanism, remote method calling from the client to the service of the server is the crucial point for the client-server communication. We have developed the idea that, if the method calling from the client on the server can be modeled by the Proxy building block's input/output pin or parameter node, we can overcome the difficulty part of modeling the client-server communication. The basic idea is that, what the client will send to the server and what the client will receive from the server will be passed by the input and output pin of the Proxy block. Depending on the type of the pin (input/output) the method signature for calling on the server will be realized in the client side. From the GWT RPC mechanism we have discussed that, at first in the client side the methods' signatures are defined, then in the Server, those methods are implemented. However, with our idea, the methods' signatures will be automatically generated from the input/output pins of the Proxy building blocks to use in the client programming.

Figure 3.11 shows an example server system that we have modeled, where we used the Proxy block for representing the client behavior.

In the system block, we can see the output pin *input_SevMsdIn* of the *GWT-GUITesting_Proxy* Proxy block passes the string in the system that the user sends to the server to make it change. This string is passed to the *changeText* operation action that performs the required change on the received string. The changed string is then passed to the client via the *response_SevMsdOut* input pin of the Proxy block. The corresponding method signature of the *input_SevMsdIn* output pin will be generated as *input_SevMsdIn(String str)* and will be realized in the client side after the implementation of this system. We can notice that, the string that the output pin passes into the server system becomes the parameter of the generated method signature to pass the user given string to the server system. The corresponding method signature of the *response_SevMsdOut* input pin of the Procy block will be generated as *String response_SevMsdOut ()*, where the received string from the server

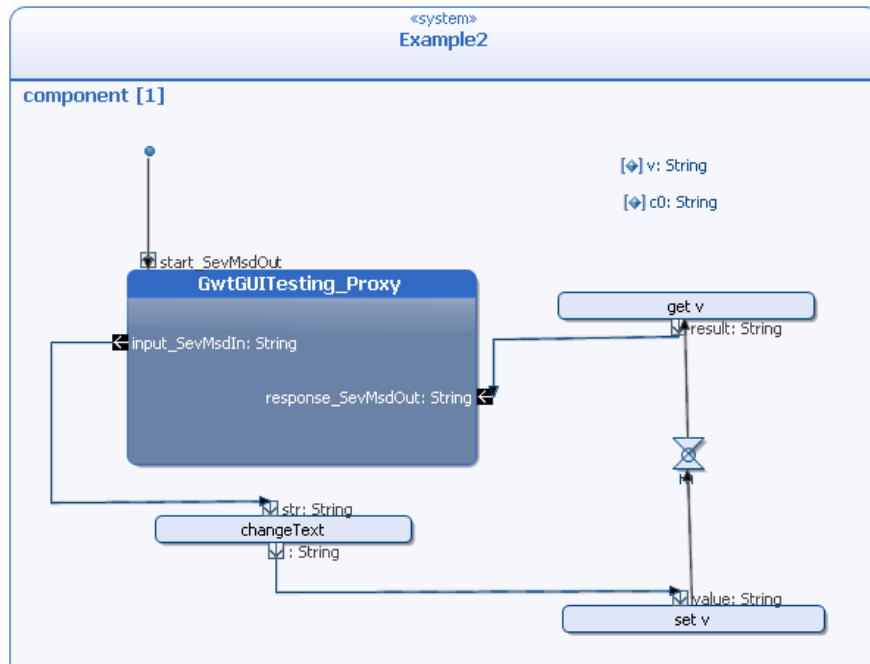


Figure 3.11: Example Server System Presenting Our Client-Server Communication Model

system becomes the return value of the method to pass it to the client. The client will call these two methods on the server to invoke the change text service of it.

To implement this model a GWT code generator is needed. So, we also have proposed the functionalities that a GWT code generator should contain in the following sub section.

3.3 GWT Code Generator

GWT code generator will perform the necessary tasks to implement an AJAX web application with GWT framework in Arctis. Figure 3.12 illustrates the role of the GWT code generator in Arctis for implementing the web application. It will create the whole GWT execution environment in Arctis to implement the AJAX web application with GWT framework. For implementing our designed model as well as for implementing the whole web application, the code generator should provide the two main functionalities. One is the GWT application layout Generation and the other is the automatic genera-

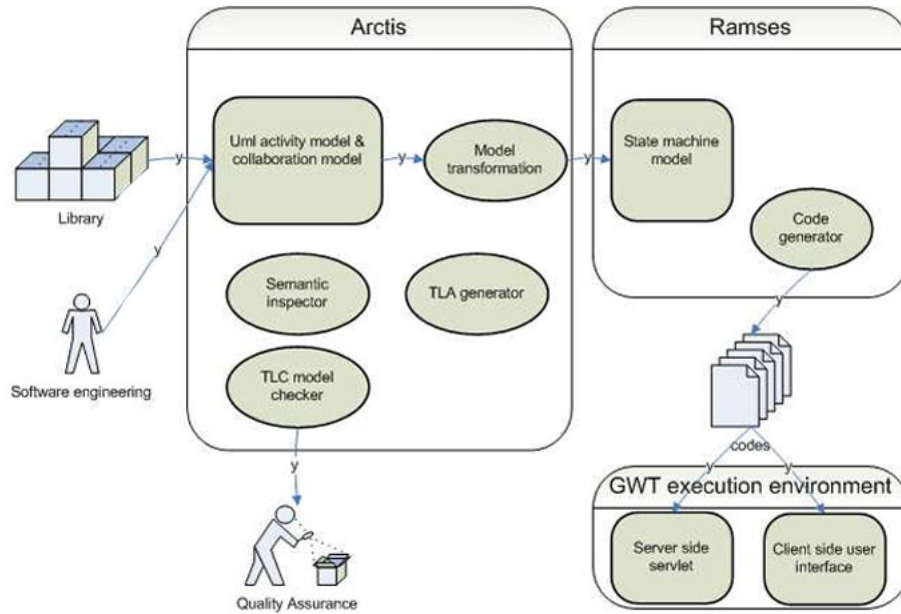


Figure 3.12: Role of GWT Code Generator in Arctis

tion of Servlet methods' signatures. In the following sub sections we describe elaborately the required functionality for a GWT code generator.

3.3.1 GWT application layout Generation

GWT relies on a specific package structure. Correctly generation of the package structure is very important for the implementation of an web application with GWT framework, as GWT relies heavily upon this package structure to infer many things about the application. The code generator should create four mandatory parts of the GWT package layout which are briefly discussed in the following subsections.

3.3.1.1 Module Descriptor

Module descriptor is an XML file which holds configuration information about the web application. In this file we can configure the information of the inherited GWT libraries used in the application, set the path of the client side code, set the path of the public resources set the entry point class, and do other advanced level configuration. The GWT code generator should

have the ability to generate this file's template creating the required specific package.

3.3.1.2 Public Resources

In this part of the package layout, the application holds the resources that will be served publicly[12]. For example, a host html page (that appears after starting the application), a CSS file, images go in this package. The GWT code generator should generate a host html templet file and a CSS template file, so that, the developers can add the necessary functionality in this file manually according to the application requirement.

3.3.1.3 Client Side Code

A specific package structure is required to hold the client side codes. It is the task of the GWT code generator to generate this structure. All the client side program files should be placed in this client package.

3.3.1.4 Server Side Code

The GWT code generator should create the specific package structure for including the server side code. The implantable code that the GWT code generator will generate from our model's state machine should also be placed as the code in this package by the code generator. Actually, the code should be placed in a servlet program in the servier side package, as we mentioned earlier that in RPC the underlying server technology is Java Servlet. Figure 3.13 shows the package layout, which should be generated by the GWT code generator.

3.3.2 Automatic Generation of the Server Methods' Signatures

For implementing our designed models the GWT code generator has to do an important task, which is the automatic generation of server methods' signatures based on our models. From the description of our client-server communication in section3.2 we can say that, the Proxy building blocks' input/output pins act as a bridge between the client and the server, as they passes data between the client and the server. Also, we have described that, the input/output pins of the Proxy building blocks will be converted into

Name	Location
Project root	<code>com/ntnu/stud/thesis/</code>
Module descriptor	<code>com/ntnu/stud/thesis/Example.gwt.xml</code>
Public resources	<code>com/ntnu/stud/thesis/war/</code>
Client-side code	<code>com/ntnu/stud/thesis/client/</code>
Server-side code	<code>com/ntnu/stud/thesis/server/</code>

Figure 3.13: Sample Locations of Different Parts of a Typical GWT Application Layout

methods' signatures for calling on the server in the client side. So, this conversion part should be done by the GWT code generator. Based on the type of the pins of the Proxy building blocks used in our model, the GWT code generator should automatically generate the methods' signatures that the client will call to invoke the service of the server or more specifically the Servlet. The GWT code generator should also automatically define the asynchronous versions of those methods. Both signatures should be placed in separate files as interfaces in the client side package by the code generator. The developer is free to perform his client side programming but should use the methods generated by the code generator to communicate with the server.

Chapter 4

Scenario Description

4.1 General Scenario

We can divide the tasks of developing web applications or web based systems into two parts- client side programming and server side programming. GWT gives the flexibility to develop the thin client meaning that most of the working functionality in web based systems will be done in the server side. This has been possible because of asynchronous communication provided by AJAX. So, in our work we have focused on the asynchronous client-server communication and server side logic modeling.

To exemplify the use case of our developed server system model the general scenario is of a web system, where the client with the help of the user interface will ask some services from the server. The server will do some processing and will provide the service to the client. Figure 4.1 shows a use case diagram of a general web system scenario.

For using all our designed building blocks, we have chosen online shopping system as a use case. In the following sections we describe our chosen scenario.

4.2 Online Shopping System as a Scenario

An Online Shopping System is a software application system that runs on a web server and provides the facilities online to customers that a real shop does. It allows the customers to search for a product in the store catalog, add a selected product to a shopping cart, place an order for the chosen

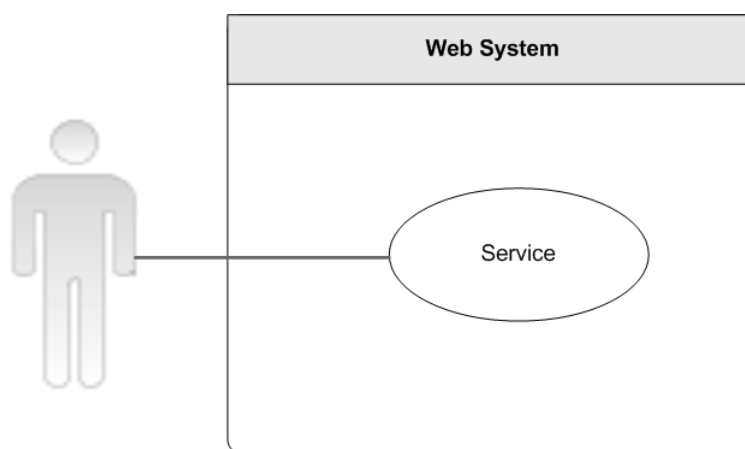


Figure 4.1: General Scenario of a Web System

products and pay online for buying the ordered products. Online Shopping System is a good example for our AJAX web application development. In an Online Shopping System, a lot of small changes in the web page are required during the shopping process of a customer. For example, before placing an order a customer may want to add several quantities of the same product in which case only the update of the price in the cart is needed in the page. In this case, a whole page refresh is not necessary. AJAX can fit here finely by providing the facility of updating only the content of the cart UI in the page.

4.2.1 Scope of the Example Scenario

The complete Online Shopping System is a complex web system and includes a storefront and administrating functionalities. The storefront provides the services for the customers that we have mentioned earlier. On the other hand, administrating functionalities allow the store administrator to manage the store for example, to add products, to set up shipping and payment options, to process orders, etc. For using our developed building blocks, we have chosen a simple Online Shopping System. Our example Online Shopping System only provides the storefront functionality and it is limited up to providing the facility for the customers to place an order for the chosen products. The services that our developed building blocks can provide through our Online Shopping System model to the customers are shown in the use case diagram in Figure 4.2.

4.2.2 Use Case Diagram of the Example Scenario

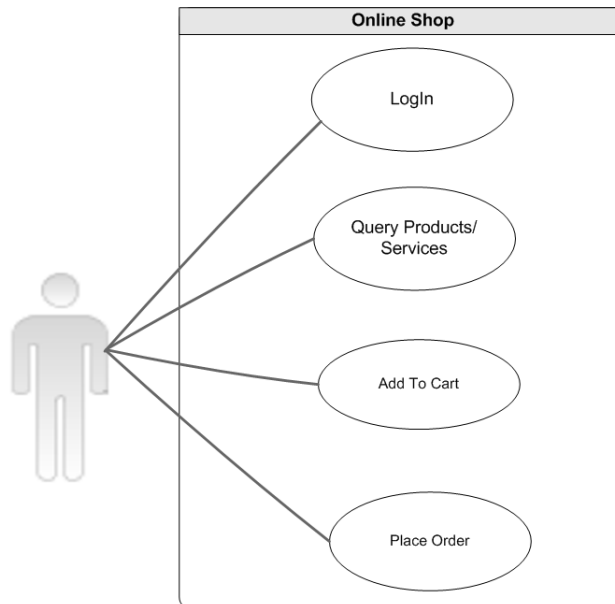


Figure 4.2: Use Case Diagram of the Example Scenario

From Figure 4.2 we can see users can log in to the Online Shopping System interacting with the LogIn service. After logging in to the system, users can find the available products in the Online Shop by clicking on the Query Product service. Having the details of available products in the shop, users can choose the products to buy from the Add to Cart service. After adding the products to the shopping cart, users can place an order for the chosen products by the Place Order service. We have modeled corresponding building blocks to provide all those services to the customer.

For the convenience of understanding the user interaction with the services in the scenario more clearly, we present an example UI in the following subsection.

4.2.3 Example UI Representing the Example Scenario

Figure 4.3 shows an example UI of our example Online Shopping System. Users can log in to the system with the help of a LogIn UI element placed in the left side of the UI. As we are using AJAX for developing the Online Shopping System, we can provide all the services that we have mentioned in

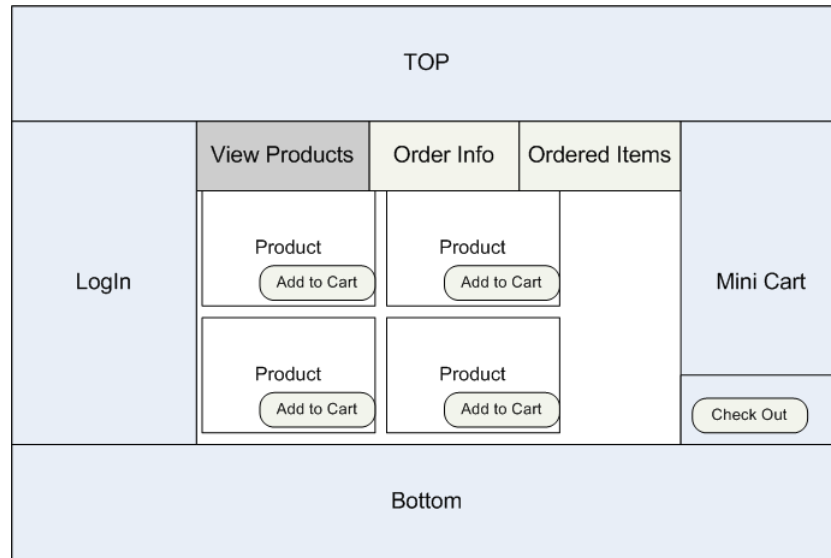


Figure 4.3: Example UI of the Example Online Shopping System

the previous section without waiting for a page refresh. The idea is that, all the services from finding the available products to placing an order can be provided by using a tab panel. Tab panel is a web UI element provided by the GWT framework, where with each tab, a corresponding panel appears. In Figure 4.3 we can see, when users click on the View Products tab, the corresponding panel appears containing the available products details. When the user wants to buy a product he can choose it by clicking the Add to Cart button as shown in Figure 4.3. The product details containing the price and the quantity in the users shopping cart can be shown in the Mini Cart panel placed at the right side of the UI. When the user wants to check out and place an order he can click on the Check Out button.

Figure 4.4 shows what happens when the user clicks on the check out button. The Ordering Info tab becomes enabled and a form appears in the corresponding panel requiring the user's shipping information, payment information and payment method choice. After filling the required information, a user can click Place Order button for placing an order for the products he had added previously in his shopping cart.

Figure 4.5 shows the status of the example UI when the user clicks on the Place Order button. Ordered Item tab becomes activated and the ordered items with the corresponding information of the order appears on the corresponding panel. The user can confirm the order that he has placed by

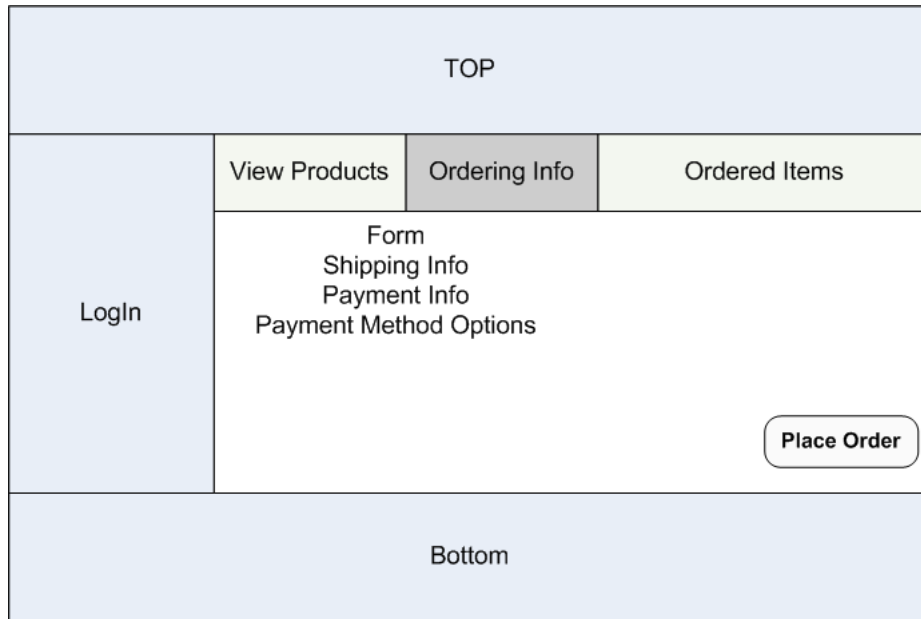


Figure 4.4: Status of the Example UI While Placing an Order

clicking the Confirm button. Departing from here the user can be directed to another service for paying online, but this is out of our scope.

4.2.4 Example Database

Every Online Shopping System maintains a relational database to store the necessary information. For our example Online Shopping System, we have designed a quite simple database. Figure 4.6 shows the summary of the database tables that we have designed for the example Online Shopping System.

Figure 4.7 shows the ER (Entity Relationship) diagram of our designed database. PK denotes the Primary Key and FK denotes the Foreign Key of the tables. The arrow represents a relationship between corresponding tables. With the help of the Primary Key, we can identify an entity in a table uniquely. The Foreign Key in a table is actually is the Primary Key of an another table. It sets a child-parent relationship between the corresponding tables and keeps the reference of the parent table in the child table. For example, if we need to know the price information of the product that is in the the child table `tbl_cart`, we can retrieve that price information from the



Figure 4.5: Status of the Example UI While Confirming an Order

parent table `tbl_product` as the `pd_id` is the column (Primary Key in the `tbl_product` and Foreign Key in the `tbl_cart`) that matches from the two tables.

Now, in the following subsections we give a brief description of each table we have designed.

4.2.4.1 `tbl_product`

In this table, we store the products' name, description, price and available quantity in the store. Here, we identify each product uniquely by giving each product a unique id.

4.2.4.2 `tbl_cart`

In this table, we store the items that a customer wants to buy in the current session. In order to do so, we have set the current session id of the customer as the Primary Key of this table. Whenever a customer adds a product in his shopping cart, we save the product id and the quantity of the product against the current session id of that customer, so that, we can retrieve later

Table Name	Table Description
tbl_product	We keep the products' information here.
tbl_cart	When the customer decides to put an item into the shopping cart, we add the item here.
tbl_order	In this table we save all orders.
tbl_orderedItem	We save the ordered items here.
tbl_user	We save the user information here.

Figure 4.6: Table Summary of the Example Database

the product and price information of the products those a customer wants to buy in the current session.

4.2.4.3 **tbl_order**

When the customer places the order, we add a new order in this table. The shipping and payment information that the customer provided after the checkout are also saved in this table. The databases have the facility to auto increment a number when a new entity will be added in a table. We can use this facility for creating order id of each order that we save in the table. We can set the starting number for the auto incrimination for example, we can set that the order id will be auto incremented starting from 1001.

4.2.4.4 **tbl_orderedItem**

We put all the ordered items here. We simply copy the items from the tbl_cart when the customer places the order and save it against the specific order id generated for the order of that specific customer.

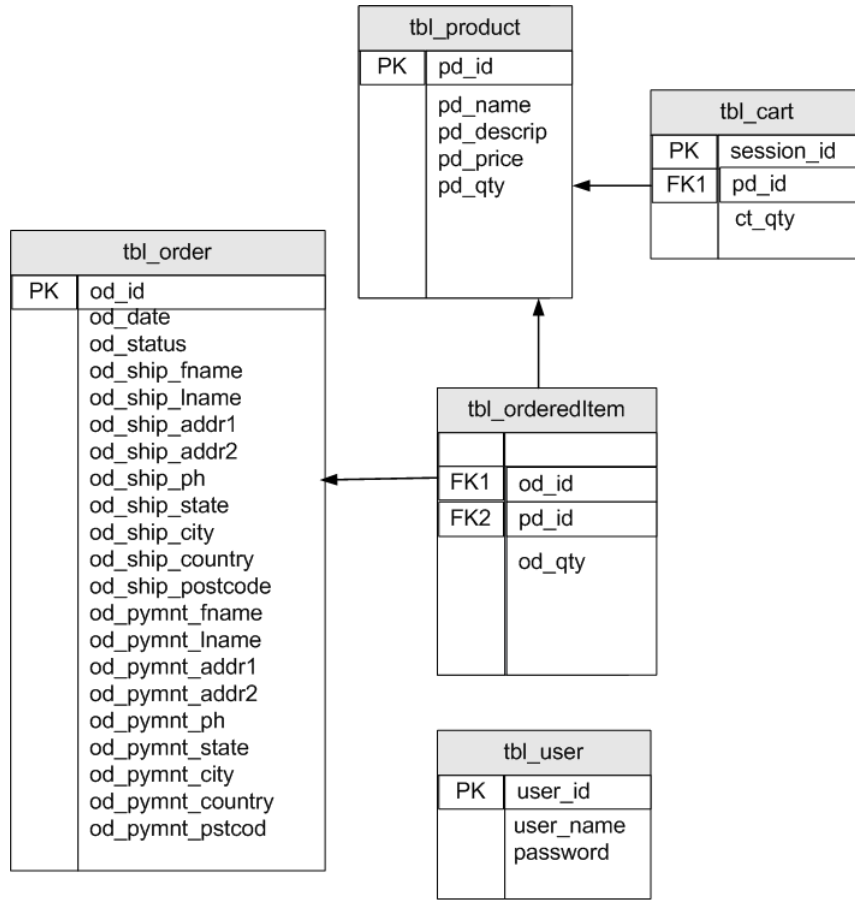


Figure 4.7: Entity Relationship Diagram of the Example Database

4.2.4.5 tbl_user

In this table, we save the users' id, name and password when the user creates an account in the system. We use this information to authentic the users who tries to log in to the system.

In the next chapter, we describe the building blocks along with their ESMs that we have modeled for our example web system. For the convenience of understanding we first describe the Online Shopping System, that we have modeled using our designed building blocks.

Chapter 5

Development of Building Blocks

5.1 System Block of the Online Shopping System

The Online Shopping System we have modeled is actually the server system of our example Online Shopping System. Figure 5.1 shows the system block of the server system that we have modeled for the example Online Shopping System.

The initial node emits a control token to start the activity of the system. In the Online Shopping System, we have described earlier that, first the customer needs to log in to the system to be able to get other services. In order to provide the log in service, the LogIn activity block that we have modeled is needed to be activated. Another thing is that, every building block we have developed does some transactions on the database or read data from the database. So, at the starting of the system, the availability of the database connection is also necessary. So, the control flow is forked into two control flows. One control flow activates the instance of the LogIn activity block by passed through its *start* pin. The other control flow activates the instance of the MySQL activity block through its *start* pin.

The MySQL activity block tries to establish a connection with the database. If the connection establishment is not successful, the control flow terminates via one of the alternative output pin *failed*. If the connection with the database is established successfully, a connection object is emitted as a data token via the other alternative output pin *success*. This connection ob-

ject is set in a universal variable *DbConnection* using the set variable control named as *set DbConnection* for later use.

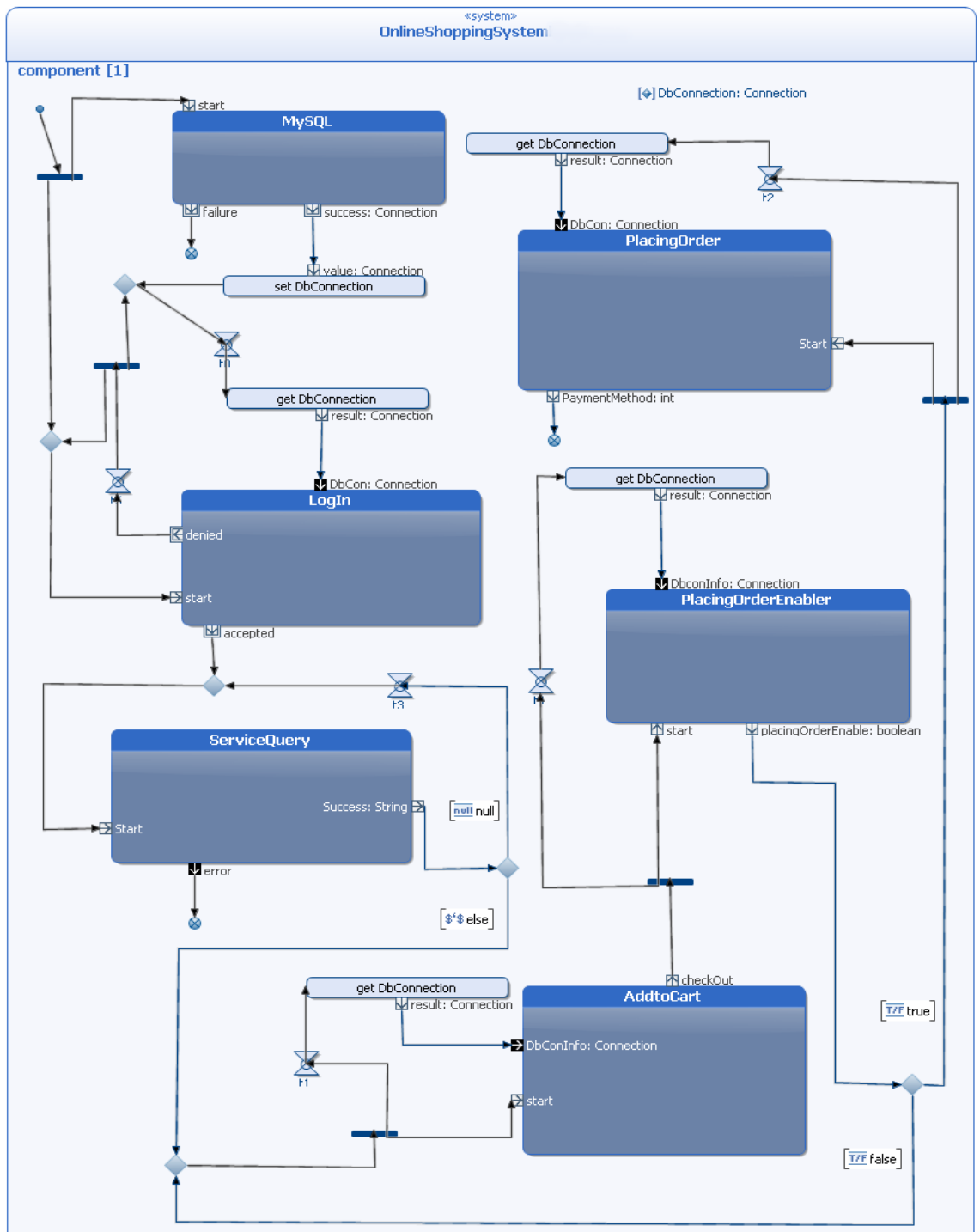


Figure 5.1: Server System Model of the Example Online Shopping System

Now, after setting the connection object, the control flow goes to a timer control. When the timer expires, the control flow goes to a read variable control named as *get DbConnection*. This control reads the value set in the *DbConnection* universal variable and passes the value as a data token via its output pin *result*. Then, the data token containing the database connection object enters the LogIn activity block via its *DbCon* streaming pin.

We have put a timer before getting the database connection object into the LogIn activity block, because we needed to let the LogIn activity block get activated before the database connection object gets available at the *DbCon* streaming input pin. The reason is that, the streaming pin of an activity block cannot pass any token via it, unless the activity block is in the active phase. In the timer, we have not set any time delay. Because of the timer, the database connection object reaches at the LogIn activity block getting out from the MySQL activity block in two action steps and this lets the LogIn activity block become activated first. Figure 5.2 clears this idea with an animation of the token flow between the MySQL and the LogIn activity block.

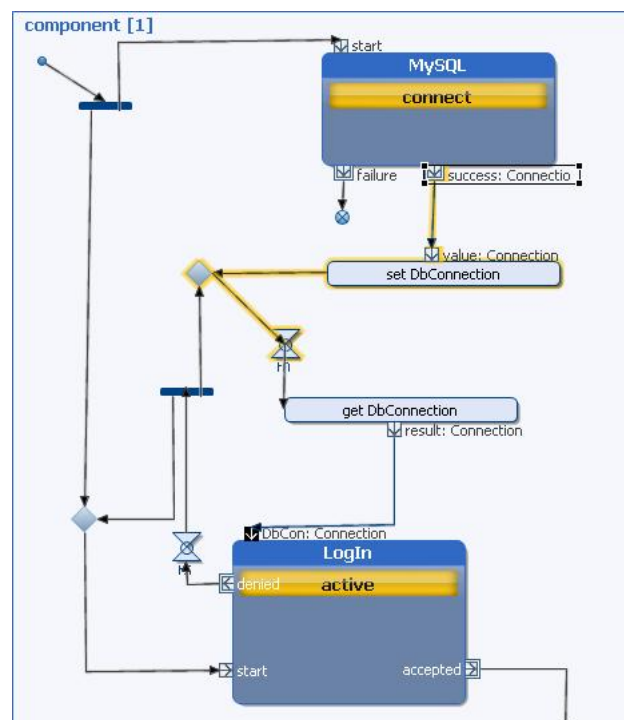


Figure 5.2: Animation of the Token Flow between the MySQL and the LogIn Activity Block

After having the database connection object, the LogIn activity block authenticates the user who tries to log in to the system. If the authentication is unsuccessful, a control token is emitted via the alternative output pin *denied*, otherwise the successful authentication emits a control token via the alternative output pin *accepted*.

If the user is denied to log in, he/she should be given the opportunity for trying to log in again. In order to make that possible, the LogIn service block should be in active phase again. So, the control token getting out from the output pin *denied* is forked to start the LogIn activity block and get the database connection object from the MySQL activity block.

The control flow that comes out via the *accepted* output pin activates the instance of the ServiceQuery activity block via its *start* pin. The ServiceQuery retrieves the names of all the available services in an existing third party Online Shopping Store named the ISIS Project Store [7] that sells services as products. We discuss more about this Online Shopping Store in chapter 6. Notice that, the available products in the existing Online Shopping Store are services, so we have named the activity block as ServiceQuery instead of ProductQuery. If any error occurs during the retrieval process, a control token is emitted via the streaming output pin *error*. At the end of the retrieval process, a string object containing the success information of the retrieval emits out via the output pin *Success* and terminates the ServiceQuery activity.

If the ServiceQuery activity block cannot retrieve the service names successfully, the string object contains null otherwise it contains a success message. If the string object contains null, the token flow goes to start the ServiceQuery activity block again. This lets the user to try for query services again. If the string object contains a success message, the token flow is forked in order to activate the AddtoCart activity block and get the database connection object successively.

The AddtoCart activity block provides the service of adding the desired items into the customer's shopping cart as many times as he wants. When the customer checks out, a control flow is passed via the *checkOut* output pin terminating the AddtoCart activity block.

Now let's think of a situation, where it is possible that, for the first time the customer tried to add an item into his shopping cart and that item cannot be added into his shopping cart, as the item is not available in the stock. Now, if the user clicks the CheckOut button, then during this time, actually his shopping cart is empty. At this stage, he should not be asked for providing his shipping and payment information, as he actually is not buying any product. We have mentioned in section 4.2.3 in the description of the example UI that,

when the `CheckOut` button is clicked, the `OrderInfo` tab is enabled to get the shipping and payment information from the user. In this case, it should not be done, as the customer's shopping cart is empty. The `ViewProducts` tab panel should remain enabled in this situation. Before enabling the `OrderInfo` tab, it should be checked whether the shopping cart of the customer is empty or not. So, in order to do that in our server system model, we have added an `PlacingOrderEnabler` activity block.

The control flow from the `checkout` output pin of the `AddtoCart` activity block is forked and one flow activates the `PlacingOrderEnabler` activity block. The other flow makes available the database connection object in the `PlacingOrderEnabler` activity block. The `PlacingOrderEnabler` block checks whether the shopping cart of the customer is empty or not. The corresponding checked result is passed as a boolean data token via the `placingOrderEnable` output pin of this block. If the data token contains a false value meaning the shopping cart of the customer is empty, the token flow goes to activate the `AddtoCart` activity block again. If the data token contains a true value, the token flow is forked in order to activate the instance of the `PlacingOrder` activity block and make available the database connection to that block.

The `PlacingOrder` activity block saves the order and creates the list of the ordered items of that order. It passes the payment method information as a data token via its `PaymentMethod` output pin. Using the payment method information passed by the `PlacingOrder` activity block, the payment activity blocks for specific payment methods can be added in the system. But, this is out of our scope, so we have terminated the token flow.

In the following sections we describe the behavior of our modeled building blocks one by one.

5.2 MySQL Building Block

Figure 5.3 shows the behavior of the MySQL building block. This building Block tries to connect with a SQL database located in the server.

This building block is activated by a control flow via its `start` input pin. Then the block tries to connect with a SQL database located in the server by the `connect` operation action. The successful connection is notified as a success event and the `success` receive signal control emits the connection object. This connection object is then passed via the alternative output pin `success`.

If any exception occurs during the connection establishment process, it is

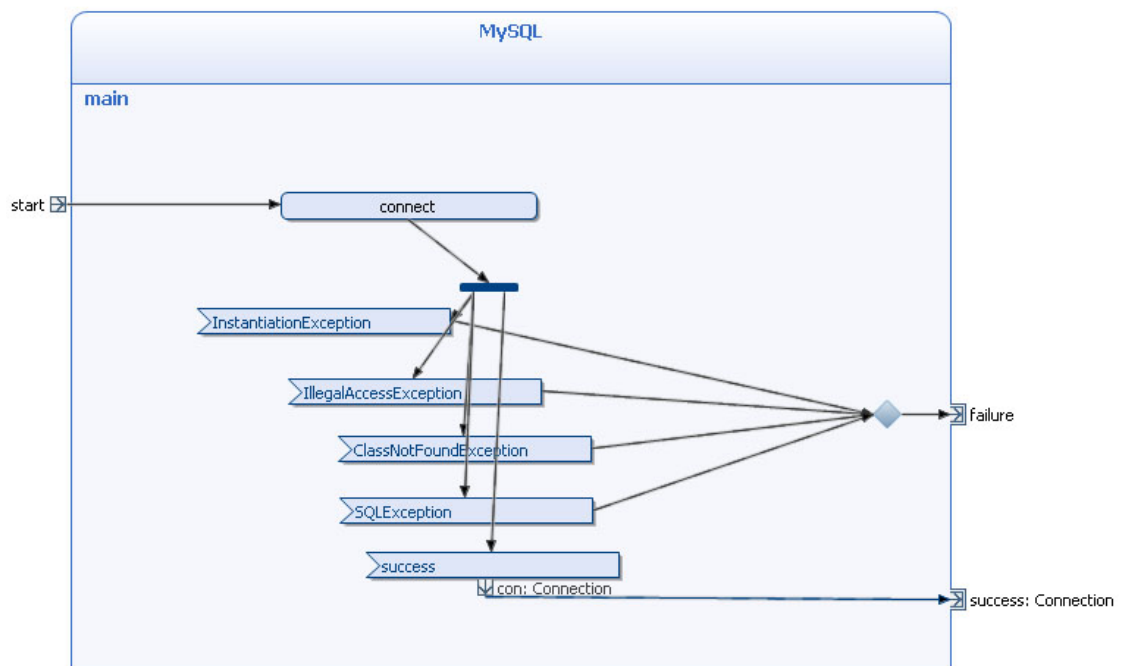


Figure 5.3: Behavior of the MySQL Building Block

notified by exception events. The corresponding receive signal control emits a control token for the specific exception event and the token is passed to the outside environment via the *failure* alternative output pin.

5.2.1 ESM of the MySQL Building Block

Figure 5.4 shows the ESM of the MySQL building block.

We have explained in section 2.3.1.5 about the purpose and semantics of ESM in details. So, here we briefly describe the ESM of the MySQL building block. The ESM describes that, after getting activated via the *start* input pin, the activity goes in a *connect* phase. From that phase the activity gets terminated by passing a token flow via either the *success* output pin or the *failure* output pin.

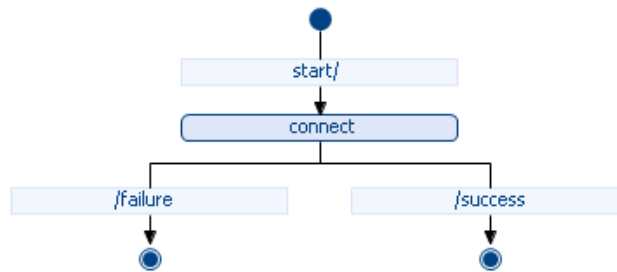


Figure 5.4: ESM of the MySQL Building Block

5.3 LogInGUI_Proxy Building Block

Figure 5.5 shows the behavior of the LogInGUI_Proxy building block.

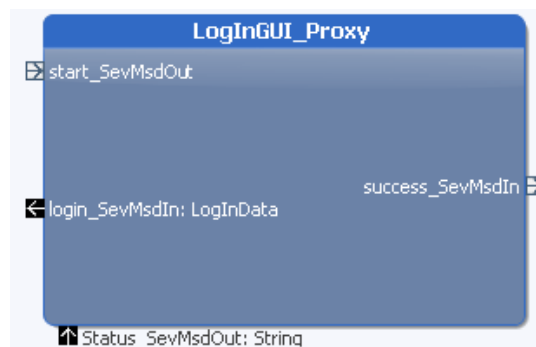


Figure 5.5: Behavior of the LogInGUI_Proxy Building Block

The LogIn GUI_Proxy building block represents the client behavior during the time period when the user tries to log in to the Online Shopping System. The activity of this building block is started by passing a control flow via its *start_SevMsdOut* pin. When the user provides the log in information, it is passed via its *login_SevMsdIn* streaming output pin to the outside environment of this building block. The outside environment of this building block will be nothing but the server system. The success or failure information of the user authentication is passed from the server to the client side via its *Status_SevMsdOut* input pin. After having the success or failure information,

a control flow is passed via its *success_SevMsdIn* terminating output pin to the surrounding server environment.

5.3.1 ESM of the LogInGUI_Proxy Building Block

Figure 5.6 shows the ESM of the LogInGUI_Proxy building block.

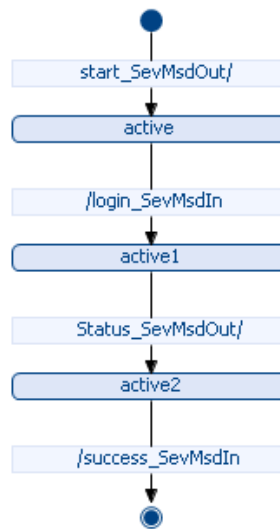


Figure 5.6: ESM of the LogInGUI_Proxy Building Block

After being activated by its *start_SevMsdOut* input pin the activity of the LogInGUI_Proxy Building Block goes in an *active* phase. When a token flow is passed via the output pin *login_SevMsdIn*, a transition from the *active* phase to an *active1* phase occurs. From the *active1* phase the activity goes in an *active2* phase, when a token enters the activity via the *Status_SevMsdOut* pin. Finally, the activity is terminated from the *active2* phase by passing a token via the *success_SevMsdIn* pin.

5.4 LogIn Building Block

Figure 5.7 shows the behavior of the LogIn building block. This building block authenticates the user who wants to log in to the Online Shopping System.

The LogIn Building Block contains the instance of the LogInGUI_Proxy building block as its inner block. The activity of the LogIn block is started, when a control flow enters via its *start* input pin. This control flow starts the activity of the LogInGUI_Proxy via its *start_SevMsdOut* input pin. Now, the LogIn activity waits for the setting up of the database connection object and the log in information of the user. The database connection object enters the LogIn activity via the streaming pin *DbCon*. When the user enters the log in information, the LogInGUI_Proxy passes that information to the LogIn activity block. When the database connection object and the user's log in information are set up in the corresponding universal variables, the activity verifies the user information against the information stored in the database in the *tbl_user* table by the *verify* operation action.

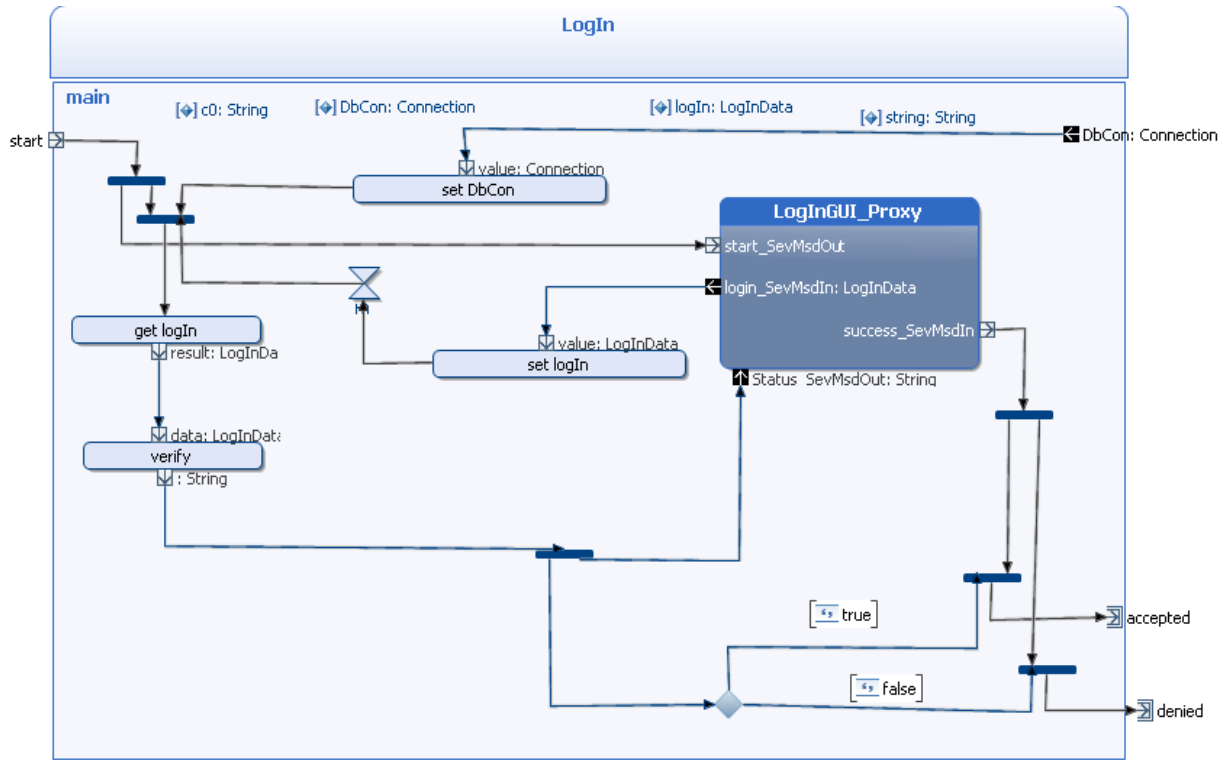


Figure 5.7: Behavior of the LogIn Building Block

The *verify* operation action passes a string object containing a true or a false value depending on the success or failure of the user authentication. This message is passed to the client side via the *Status_SevMsdOut* pin of the LogInGUI_Proxy. When this status message is successfully received in the

client side, a control flow is passed via the *accepted* or *denied* output pin of the LogIn activity depending on the user authentication success or failure and the activity terminates.

5.4.1 ESM of the LogIn Building Block

Figure 5.8 shows the ESM of the LogIn building block.

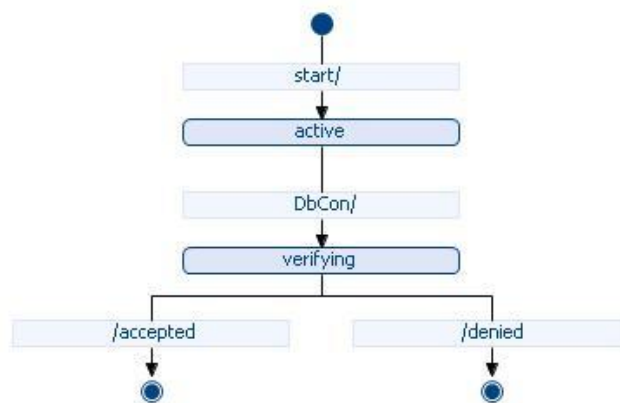


Figure 5.8: ESM of the LogIn Building Block

The activity of the LogIn building block is started by a control flow passed via the *start* input pin and the activity goes in an *active* phase. When the database connection object is passed into the activity via the *DbCon* input pin the activity goes into a *verifying* phase from the *active* phase. From the *verifying* phase the activity can be terminated by passing a token via either the *accepted* output pin or the *denied* output pin.

5.5 ServiceQuery_Proxy Building Block

Figure 5.9 shows the behavior of the ServiceQuery_Proxy building block.

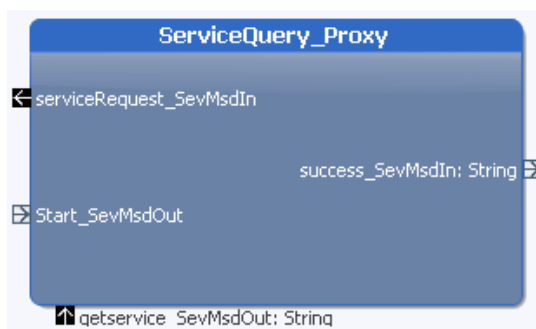


Figure 5.9: Behavior of the ServiceQuery_Proxy Building Block

The ServiceQuery_Proxy building block represents the client behavior when the user wants to know the name of the available services in the Online Shopping Store from the server. The ServiceQuery_Proxy is started via the *start_SevMsdOut* input pin. When the client requests the server for the available services a control flow is passed via the *serviceRequest_SevMsdIn* output pin of the ServiceQuery_Proxy building block. When the server returns the available service names, it is passed to the client as a string via the *getservice_SevMsdOut* pin of the ServiceQuery_Proxy. After the reception of this information, the client sends a message to the server containing a success message or a null depending on the successful reception of the service names or not via the *success_SevMsdIn* pin of the ServiceQuery_Proxy.

5.5.1 ESM of the ServiceQuery_Proxy Building Block

Figure 5.10 shows the ESM of the ServiceQuery_Proxy building block.

After getting started via the *sart_SevMsdOut* pin, the activity of the ServiceQuery_Proxy goes in an *active* phase. When a control flow is passed via the *servicRequest_SevMsdIn* output pin to the outside of the ServiceQuery_Proxy, a transition of the activity from the *active* phase to an *active1* phase happens. When the service names are passed as a string from the server environment to the client via the *getservice_SevMsdOut* pin, the activity goes in an *active2* phase from the *active1* phase. Finally, a control flow passing via the *success_SevMsdIn* pin terminates the activity of the ServiceQuery_Proxy.

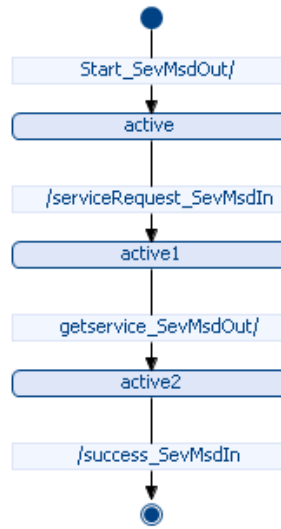


Figure 5.10: ESM of the ServiceQuery_Proxy building block

5.6 ServiceQuery Building Block

Figure 5.11 shows the behavior of the ServiceQuery building block.

The activity of the ServiceQuery is started by a control flow passed via the *Start* pin. This control flow also starts the activity of the ServiceQuery_Proxy building block. Then the activity of the ServiceQuery waits for the request for the service names from the client. When the client requests to the server asking the service names stored in the Online Shopping Store, the activity of the ServiceQuery tries to connect with the data storage of the Online Shopping Store by the *connect* operation action. The successful connection is notified as a *connection_successful* event. Then the activity tries to fetch the available service names from the data storage of the Online Shopping Store by the *getService* operation action. If the service names are fetched successfully, it is notified as a *get_successful* event in the activity and the activity pass the service names to the client via the *getservice_SevMsdOut* pin of the ServiceQuery_Proxy.

Exceptions may occur during the connection establishment with the data storage of the Online Shopping Store. Also, exceptions may occur when the service names are being fetched from the data storage. These exceptions are notified as corresponding exception events and a control flow is passed to the outside environment of the activity via the *error* streaming output pin. When any exception occurs, the service names cannot be fetched from the data

storage of the Online Shopping Store. In this situation, the client will receive a null value via the *getservice_SevMsdOut* pin of the *ServiceQuery_Proxy*. After receiving the information of the service names, the client will send the success or failure message to the server via the *success_SevMsdIn* pin of the *ServiceQuery_Proxy*. The token flow getting out from the *success_SevMsdIn* pin of the *ServiceQuery_Proxy* terminates the activity of the *ServiceQuery* block passing via its *Success* pin.

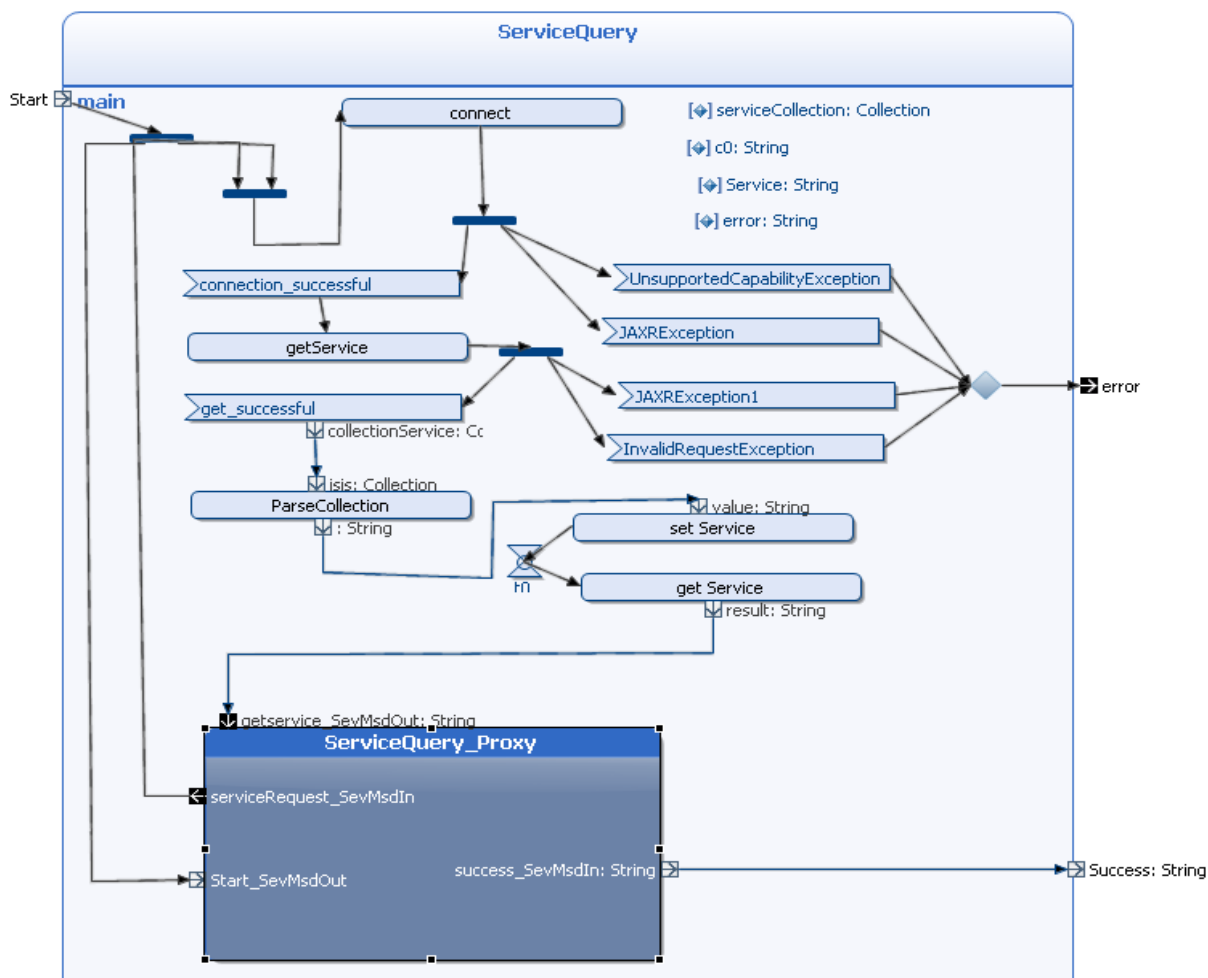


Figure 5.11: Behavior of the ServiceQuery Building Block

5.6.1 ESM of the ServiceQuery Building Block

Figure 5.12 shows the ESM of the ServiceQuery building block.

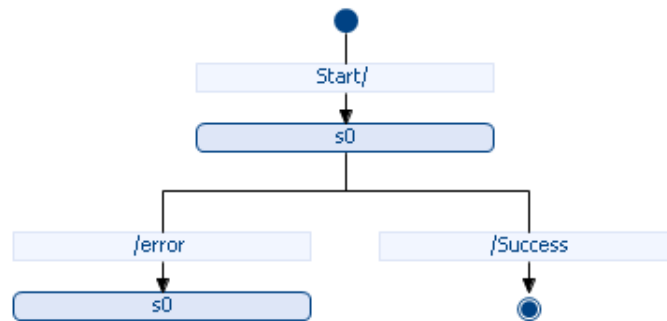


Figure 5.12: ESM of the ServiceQuery Building Block

After being started via the *Start* pin, the activity of the ServiceQuery goes in a *s0* phase. When any exception occurs, a control flow is passed via the error output pin and the activity goes again in the *s0* phase. The activity gets terminated when a token flow is passed via the *Success* output pin of it.

5.7 AddtoCartGUI_Proxy Building Block

Figure 5.13 shows the behavior of the AddtoCartGUI_Proxy building block.

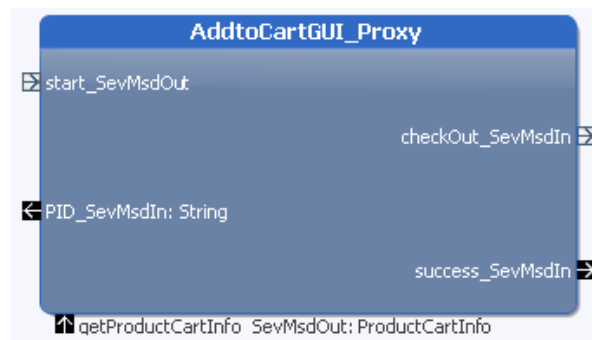


Figure 5.13: Behavior of the AddtoCartGUI_Proxy Building Block

The AddtoCartGUI_Proxy represents the client behavior when the customer wants to add the desired product item into his shopping cart. The activity of the AddtoCartGUI_Proxy is started when a control flow is passed via the *start_SevMsdOut* pin. The product id number is passed to the server by the

client via the *PID_SevMsdIn* output pin of the AddtoCartGUI_Proxy. The product information that has been added in the customer's shopping cart is returned by the server to the client via the *getProductCartInfo_SevMsdOut* pin of the AddtoCart_Proxy. When the product information is received successfully by the client, a control flow is passed via the *success_SevMsdIn* output pin of this block to its outside environment. When the customer checks out of adding products into his shopping cart, a control flow is passed via the *checkOut_SevMsdIn* output pin of the AddtoCart_Proxy and it gets terminated.

5.7.1 ESM of the AddtoCartGUI_Proxy Building Block

Figure 5.14 shows the ESM of the AddtoCartGUI_Proxy building block.

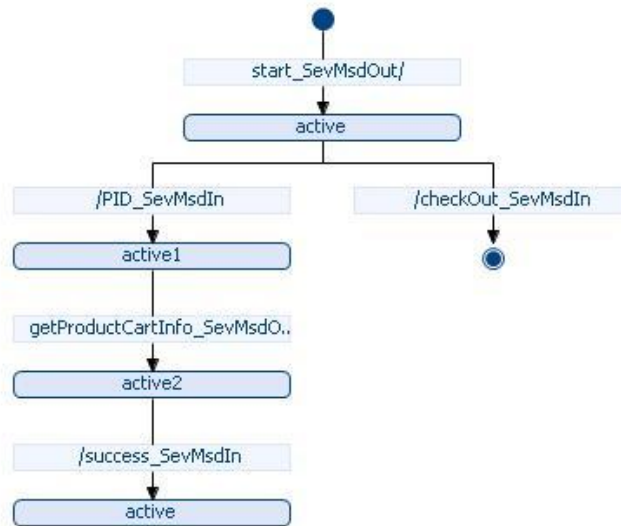


Figure 5.14: ESM of the AddtoCartGUI_Proxy Building Block

The AddtoCartGUI_Proxy is started by a control flow passing via its *start_SevMsdOut* pin and the activity goes in an *active* phase. From this state the activity of the AddtoCartGUI_Proxy goes in an *active1* phase, when a product id number is passed via its *PID_SevMsdIn* output pin. The activity then goes in an *active2* phase, when the product information added to the shopping cart is passed via the *getProductCartInfo_SevMsdOut* input pin. After the successful reception of the product information by the client, a control

flow is passed via the *success_SevMsdIn* output pin of the *AddtoCart_Proxy* and the activity goes in the *active* phase again to let the customer add more products if he wants. The activity of the *AddtoCartGUI_Proxy* can be terminated by passing a control flow via its *checkOut_SevMsdIn* output pin.

5.8 AddtoCart Building Block

Figure 5.15 shows the behavior of the *AddtoCart* building block.

The activity of the *AddtoCart* building block is started when a control flow is passed via its *start* pin. When the database connection object is obtained from the surrounding environment via the *DbConInfo* pin, it is set in a universal variable *DbConInfo*. After that, a control flow activates the instance of the *AddtoCartGUI_Proxy*. When the product id from the client is obtained in the activity of the *AddtoCart* block via the *PID_SevMsdIn* pin of the *AddtoCartGUI_Proxy*, the activity checks the quantity of the product by the *checkInStock* operation action. The *checkInStock* operation action checks the quantity of the product searching in the *tbl_product* table of the database against the given product id. The quantity of the product is returned as a data token by the *checkInStock* operation action in the activity.

If there is any product in the stock, the activity then checks whether the product item already exists in the shopping cart of that customer for that session by the *alreadyInCart* operation action. The *alreadyInCart* operation action checks the existence of the product item in the *tbl_cart* table of the database searching against the current session id of the customer and the given product id. If there is no such product item already in the shopping cart of that customer, the product item is added against the current session id of the customer in the *tbl_cart* table of the database by the *insertIntoCart* operation action. If the product item already exists in the shopping cart of the customer, it means the customer wants to buy one more product of the same item. So, the quantity is updated in the *tbl_cart* table of that product item for that customer by the *updateQty* operation action.

If the product is not available in the stock, a zero value is set in a universal variable *qtyInStock* by a set variable control named *set qtyInStock*. Now, the activity of the *AddtoCart* building block passes the product information added in the shopping cart of the customer to the client side by the *getProductInfo* operation action via the *getProductCartInfo_SevMsdOut* pin of the *AddtoCartGUI_Proxy*.

The *getProductInfo* operation action reads the quantity of the given product

item from the `tbl_cart` table in the database searching against the current session id of the customer. Getting the price information for that product item from the `tbl_product` table in the database, the `getProductInfo` operation action calculates the subtotal price of that product item. A data token containing the product id, its unit price, quantity and the subtotal price is returned by the `getProductInfo` operation action.

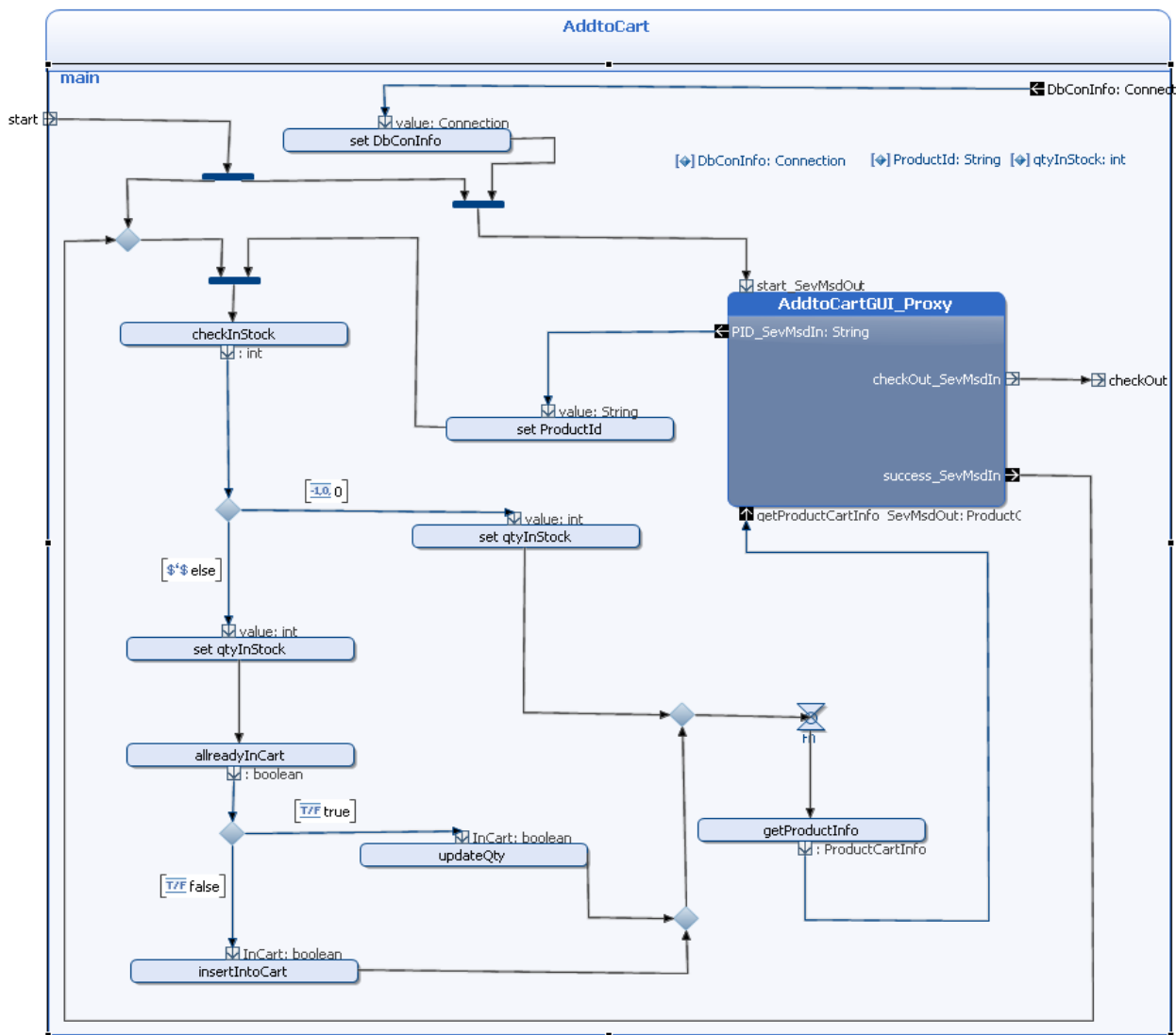


Figure 5.15: Behavior of the AddtoCart Building Block

After the successful reception of this product information by the client side, a

control flow is passed in the activity of the AddtoCart via the *success_SevMsdIn* output pin of the AddtoCartGUI_Proxy building block. This control flow brings the activity of the AddtoCart building block in a previous state, from where the activity of the AddtoCart block can perform the activities again for adding a new product item. This lets the customer to add in his shopping cart as many products as he wants until he checks out. When the customer checks out, a control is passed via the *checkOut_SevMsdIn* output pin of the AddtoCartGUI_Proxy and this control flow terminates the activity of the AddtoCart passing via its *checkOut* output pin.

5.8.1 ESM of the AddtoCart Building Block

Figure 5.16 shows the ESM of the AddtoCart building block.

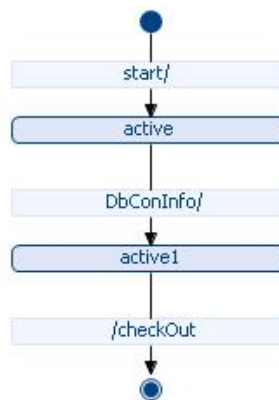


Figure 5.16: ESM of the AddtoCart Building Block

The activity of the AddtoCart building block is started via its *start* pin. Then the activity goes in an *active* phase and waits for a token flow to be passed via the *DbConInfo* input pin. When a token flow is passed via the *DbConInfo* input pin, the activity goes in an *active1* phase. In this state, the activity performs the adding of the product items in a customer's shopping cart as many times as he wants. The activity goes into the termination by a control flow passed via its *checkOut* output pin.

5.9 OrderInfoEnabler_Proxy Building Block

Figure 5.17 shows the behavior of the OrderInfoEnabler_Proxy building block.

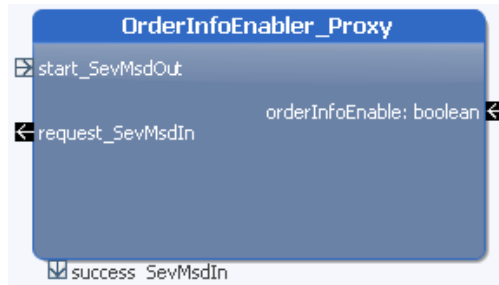


Figure 5.17: Behavior of the OrderInfoEnabler_Proxy Building Block

The OrderInfoEnabler_Proxy building block represents the client side behavior when the customer clicks the CheckOut button in order to place an order for the products he wants to buy. The activity of the OrderInfoEnabler_Proxy gets started by a control flow passed via the *start_SevMsdOut* pin of this block. When the client side sends a request to the server to know whether the OrderInfo tab will be enabled for collecting the shipping and the payment information from the user or not, a control flow is passed via the *request_SevMsdIn* output pin of the OrderInfoEnabler_Proxy to the server. When the server provides a reply with a true or false value, a data token is passed via the *orderInfoEnable_SevMsdOut* pin of the OrderInfoEnabler_Proxy to the client. When the client receives this information successfully, a control flow is passed to the server via the *success_SevMsdIn* pin of the OrderInfoEnabler_Proxy.

5.9.1 ESM of the OrderInfoEnabler_Proxy Building Block

Figure 5.18 shows the ESM of the OrderInfoEnabler_Proxy building block.

The activity of the OrderInfoEnabler_Proxy is started via its *start_SevMsdOut* input pin. Then the activity goes in an *active* phase. When a token flows via the *request_SevMsdIn* input pin of the OrderInfoEnabler_Proxy, it goes in an *active1* phase from the *active* phase. From this phase the activity goes in an *active2* phase when a data token flows via its *orderInfoEnable_SevMsdOut*

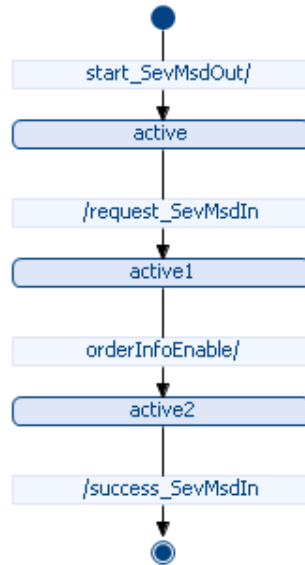


Figure 5.18: ESM of the OrderInfoEnabler_Proxy Building Block

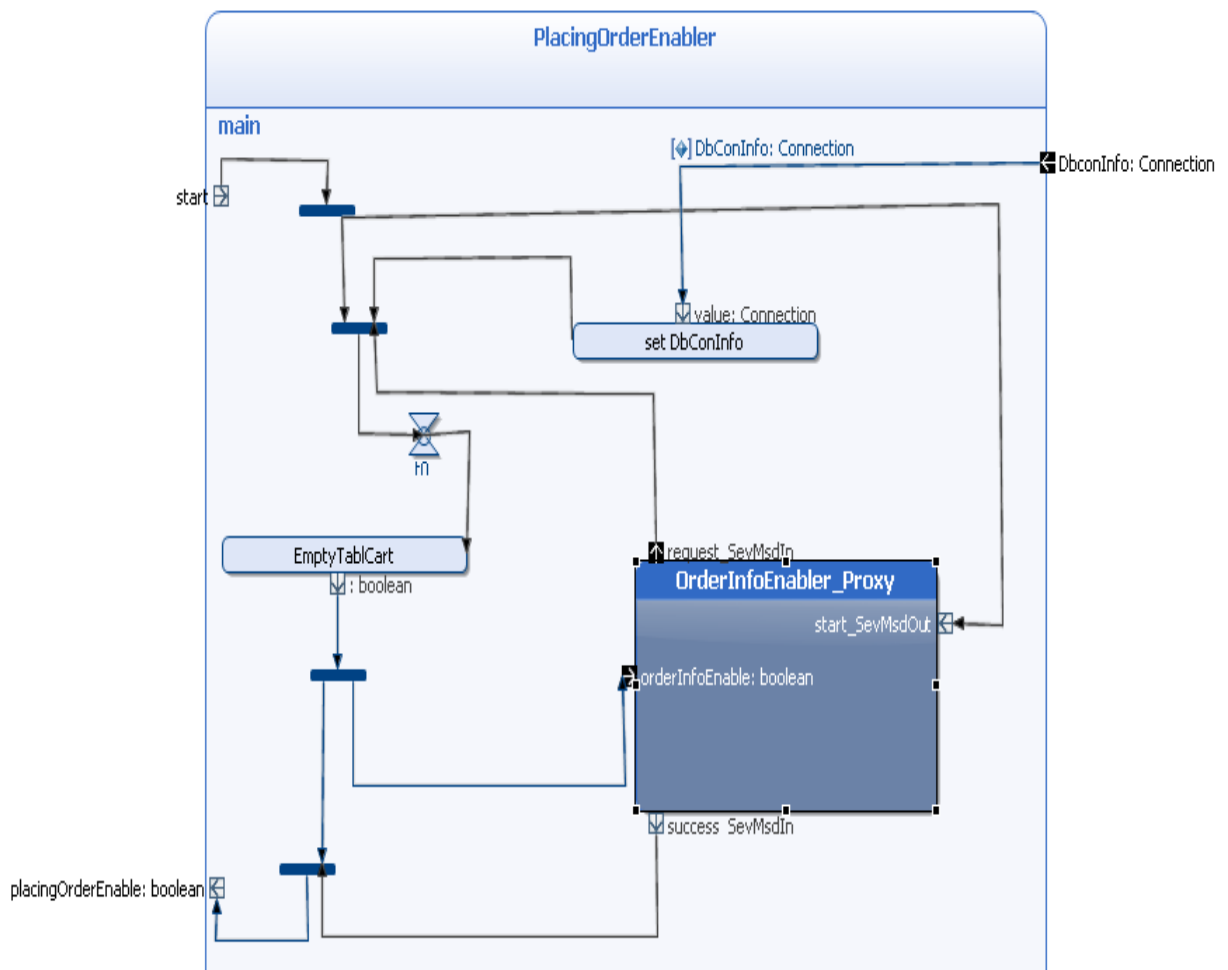
input pin. Finally, the activity can be terminated by passing a token flow via its *success_SevMsdIn* output pin.

5.10 PlacingOrderEnabler Building Block

Figure 5.19 shows the behavior of the PlacingOrderEnabler building block.

The PlacingOrderEnabler building block is activated when a control flows via its *start* pin. This control flow activates the instance of the OrderInfoEnabler_Proxy building block. When the database connection object is obtained and the client sends a request by passing a control flow via the *request_SevMsdIn* pin of the OrderInfoEnabler_Proxy, the activity of the PlacingOrderEnabler checks in the database whether the customer's shopping cart is empty or not by the *EmptyTablCart* operation action.

The *EmptyTablCart* operation action checks in the *tbl_cart* table in the database whether there is any entry with the customer's current session id or not. Depending on that checking a true or a false value is returned by the *EmptyTablCart* operation action. This information is sent to the client side by the activity via the *orderInfoEnable_SevMsdOut* pin of the OrderInfoEnabler_Proxy. When a control flow is passed via the *success_SevMsdIn*

Figure 5.19: Behavior of the `PlacingOrderEnabler` Building Block

pin of the `OrderInfoEnabler`, the activity of the `PlacingOrderEnabler` gets terminated by passing a data token via its `placingOrderEnable` output pin containing a true or a false value returned from the `EmptyTablCart` operation action.

5.10.1 ESM of the `PlacingOrderEnabler` Building Block

Figure 5.20 shows the ESM of the `PlacingOrderEnabler` building block.

After getting activated via the `start` input pin the activity of the `PlacingOrderEnabler` goes in an *active* phase. In this phase, the activity waits for

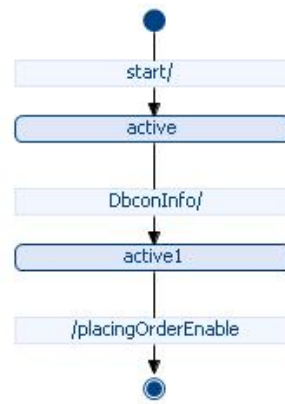


Figure 5.20: ESM of the PlacingOrderEnabler Building Block

a database connection object to be passed via the *DbConInfo* pin in order to go in an *active1* phase. From the *active1* phase, the activity gets terminated when a data token is passed via its *placingOrderEnable* output pin.

5.11 OrderingUI_Proxy Building Block

Figure 5.21 shows the behavior of the OrderingUI_Proxy building block.



Figure 5.21: Behavior of the OrderingUI_Proxy Building Block

The OrderingUI_Proxy building block represents the client side behavior

when the customer fills up the shipping and payment information and confirms the order he has placed. The activity of the OrderingUI_Proxy building block is started via its *start_SevMsdOut* pin. When the customer fills up the shipping and payment information, the information is sent out to the server via the *OrderInfo_SevMsdIn* pin of the OrderingUI_Proxy. When the server returns the created list of the ordered items along with other order information to the client side, a data token containing the corresponding information is passed via the *OrderedItem_SevMsdOut* pin of the OrderingUI_Proxy to the client side. When the customer confirms the placed order, a control flow is passed via the *Confirmed_SevMsdIn* output pin of the OrderingUI_Proxy to the server.

5.11.1 ESM of the OrderingUI_Proxy Building Block

Figure 5.22 shows the ESM of the OrderingUI_Proxy building block.

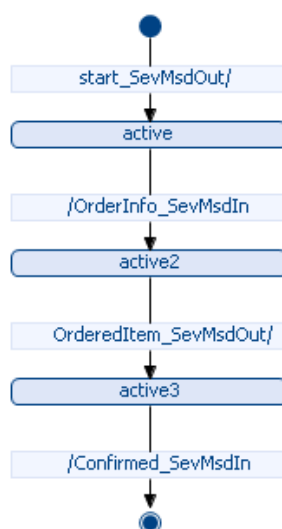


Figure 5.22: ESM of the OrderingUI_Proxy Building Block

The activity of the OrderingUI_Proxy goes in an *active* phase after getting started via the *start_SevMsdOut* pin. When a token flow is passed via the *OrderInfo_SevMsdIn* output pin of the OrderingUI_Proxy, its activity goes in an *active2* phase from the *active* phase. A transition of the activity from the *active2* phase to an *active3* phase happens when a token flows via the *OrderedItem_SevMsdOut* pin. The activity is terminated by a token flow passing via its *Confirmed_SevMsdIn* output pin.

5.12 PlacingOrder Building Block

Figure 5.23 shows the behavior of the PlacingOrder building block.

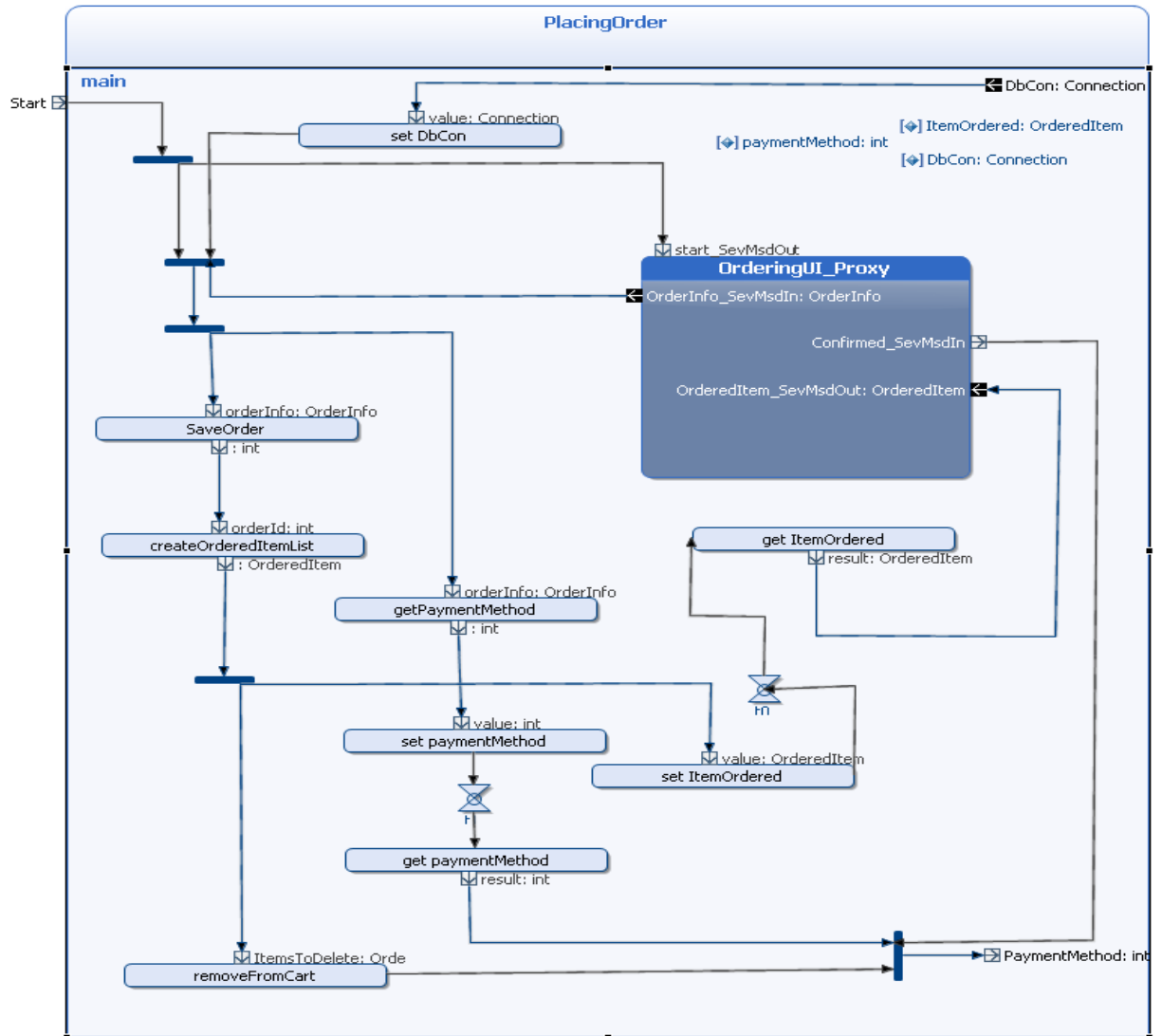


Figure 5.23: Behavior of the PlacingOrder Building Block

A control flow passing via the *Start* pin of the PlacingOrder building block activates this block. This control flow then activates the OrderingUI_Proxy being passed via its *start_SevMsdOut* pin. When the database connection object is obtained from the outside environment and the shipping and payment

information is obtained from the client side via the *OrderInfo_SevMsdIn* pin of the *OrderingUI_Proxy*, the activity of the *PlacingOrder* building block saves the order by the *SaveOrder* operation action.

The *SaveOrder* operation action saves the shipping information and the payment information as an order entry in the *tbl_order* table in the database. The database automatically generates a unique order id for a new order in its *tbl_order* table. After saving the order in the database, the *SaveOrder* operation action returns the order id in the activity. Using this order id the activity saves the ordered items in the database in the *tbl_orderedItem* table and creates a list of the ordered items by the *createOrderedItemList* operation action.

The *createOrderedItemList* operation action creates an entry against the given order id in the *tbl_orderedItem* table in the database for each product item in the shopping cart of the customer for the current session. The *createOrderedItemList* creates the ordered item's list where each ordered item data object contains the ordered product id and its subtotal price. The *createOrderedItemList* operation action also calculates the total price of the ordered items. The *createOrderedItemList* operation action returns all these information along with the order id encapsulating in a data object. The *createOrderedItemList* operation action performs all these tasks getting the necessary information from the *tbl_cart*, *tbl_product*, *tbl_order* and the *tbl_orderedItem* tables in the database.

The data object returned by the *createOrderedItemList* operation action is passed to the client side via the *OrderedItem_SevMsdOut* pin of the *OrderingUI_Proxy*. At the same time the products from the shopping cart of the customer for the current session are removed via the *removeFromCart* operation action. The *removeFromCart* operation action removes the entries from the *tbl_cart* table in the database those match the current session id of the customer.

Now, when a control flow is passed via the *Confirmed_SevMsdIn* output pin of the *OrderingUI_Proxy*, and the payment method information is obtained from the customer provided payment information, the payment method information is passed as a data token via the *paymentMethod* output pin of the *PlacingOrder* activity. This causes the termination of this activity.

5.12.1 ESM of the *PlacingOrder* Building Block

Figure 5.24 shows the ESM of the *PlacingOrder* building block.



Figure 5.24: ESM of the PlacingOrder Building Block

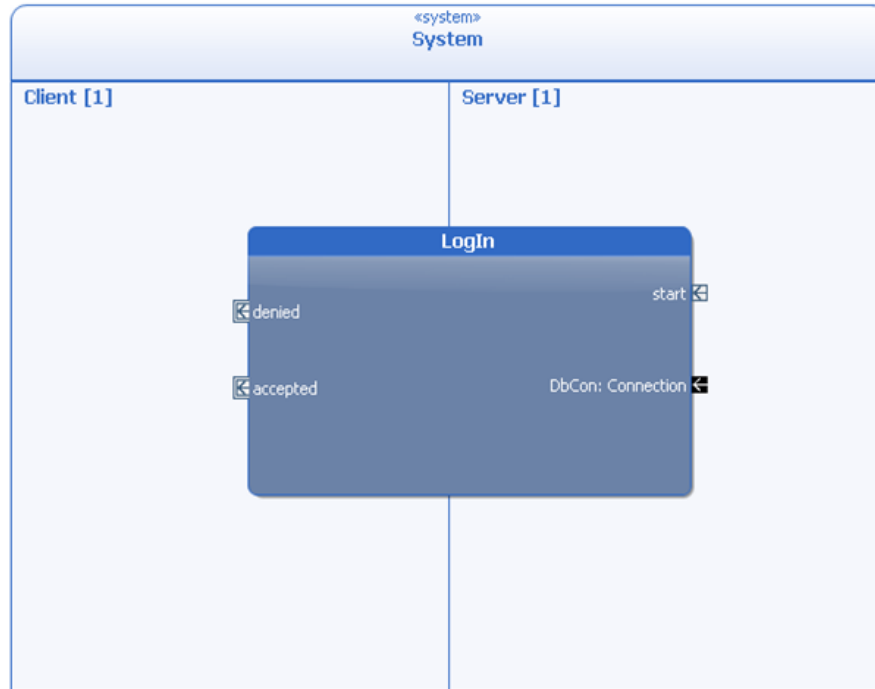
The activity of the PlacingOrder building block is started via the *Start* input pin and goes in an *active* phase. When the database connection object is entered via the *DbCon* input pin, the activity goes in an *active2* phase from the *active* phase. Finally, the activity gets terminated, when a token is passed via its *PaymentMethod* output pin.

5.13 Our Service Specification in the Perspective of the SPACE Method

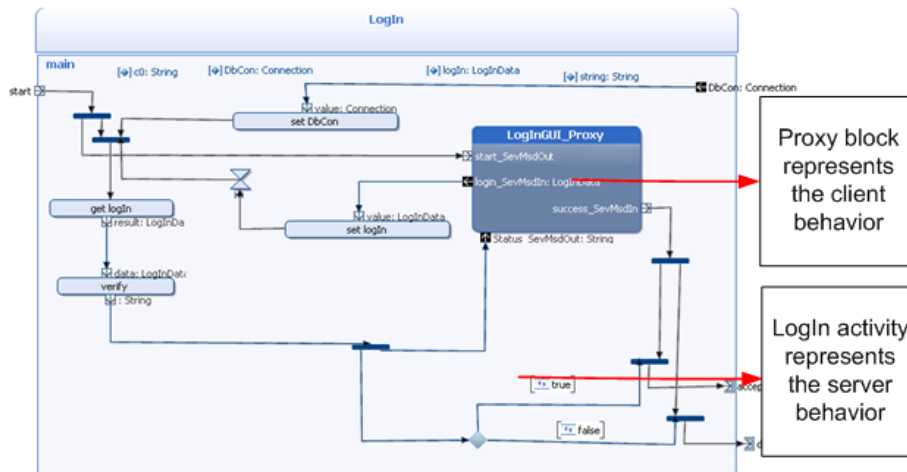
SPACE method specifies services as collaborations among the participating system components. However, collaborative models are not the only feature provided by the SPACE method. According to SPACE method, systems can be specified as a hierarchy of building blocks. We have not designed collaborative models for the specification of our services. Rather, we have designed our services as a hierarchy of building blocks. Figure 5.25 shows the difference between these two specification methods. In the perspective of a web system, the participating system components are the client and the server. If we want to specify the services provided by the server as collaborations among the client and the server, the design would have been like Figure 5.25 (a).

However, our service model is a building block that contains the client be-

havior as its inner block. The reason behind such modeling is that, we are using the Proxy building block to represent the client behavior and the Proxy block itself represents the proxy of an outside system. In our model we have counted the local system as the server system and the Proxy represents the proxy of a client system.



a)



b)

Figure 5.25: (a)Service Specification as a Collaboration among the Participating System Components, (b)Service Specification as a Hierarchy of Building Blocks

Chapter 6

Implementation

6.1 Model Checking

Formal analysis based on automatic model checking is an essential step to ensure the soundness of the designed model. This analysis on the system specification allows removing of the erroneous situations that may create problems during the implementation of the modeled system. So, before approaching towards automatic model transformation and implementation step, we have performed the analysis on our specified system model.

With SPACE method and its tool suit Arctis, we can verify a system incrementally, since encapsulating the activity blocks with ESMs allows analyzing the activities separately [20]. we have discussed earlier in section 2.3.2 and 2.4 that, Arctis supports formal analysis on specifications or models providing an automated model checker. With this advanced automated model checker supported by Arctis, we have performed analysis on each of our activity block separately. After making sure the soundness of each of our building block through model checking, we have analyzed our specified system. With the help of the model checker, we have also ensured the soundness of our specified system.

In the following sub sections we present a brief view of our analysis.

6.1.1 Analysis on Building Blocks

You can notice in LogIn activity block in section 5.4 in Figure 5.7 that, we have put a timer in the activity after having the log in information from the client via the *login_SevMsdIn* output pin of the LogInGUI_Proxy. Previ-

ously, we had not put any timer and during the analysis of the LogIn building block we have been notified with an error situation. The model checker notified us that, the ESM of the inner block has been harmed. The error situation is shown in Figure 6.1.

The ESM of the LogInGUI_Proxy in section 5.3.1 in Figure 5.6 shows that, when the activity of the LogInGUI_Proxy is in an *active* phase, a token flow can only traverse the *login_SevMsdIn* pin of this activity. However, without the timer in the LogIn activity block which we have put, the token flow also traverses the *Status_SevMsdIn* input pin of the LogInGUI_Proxy. This situation harms the ESM of the LogInGUI_Proxy activity block.

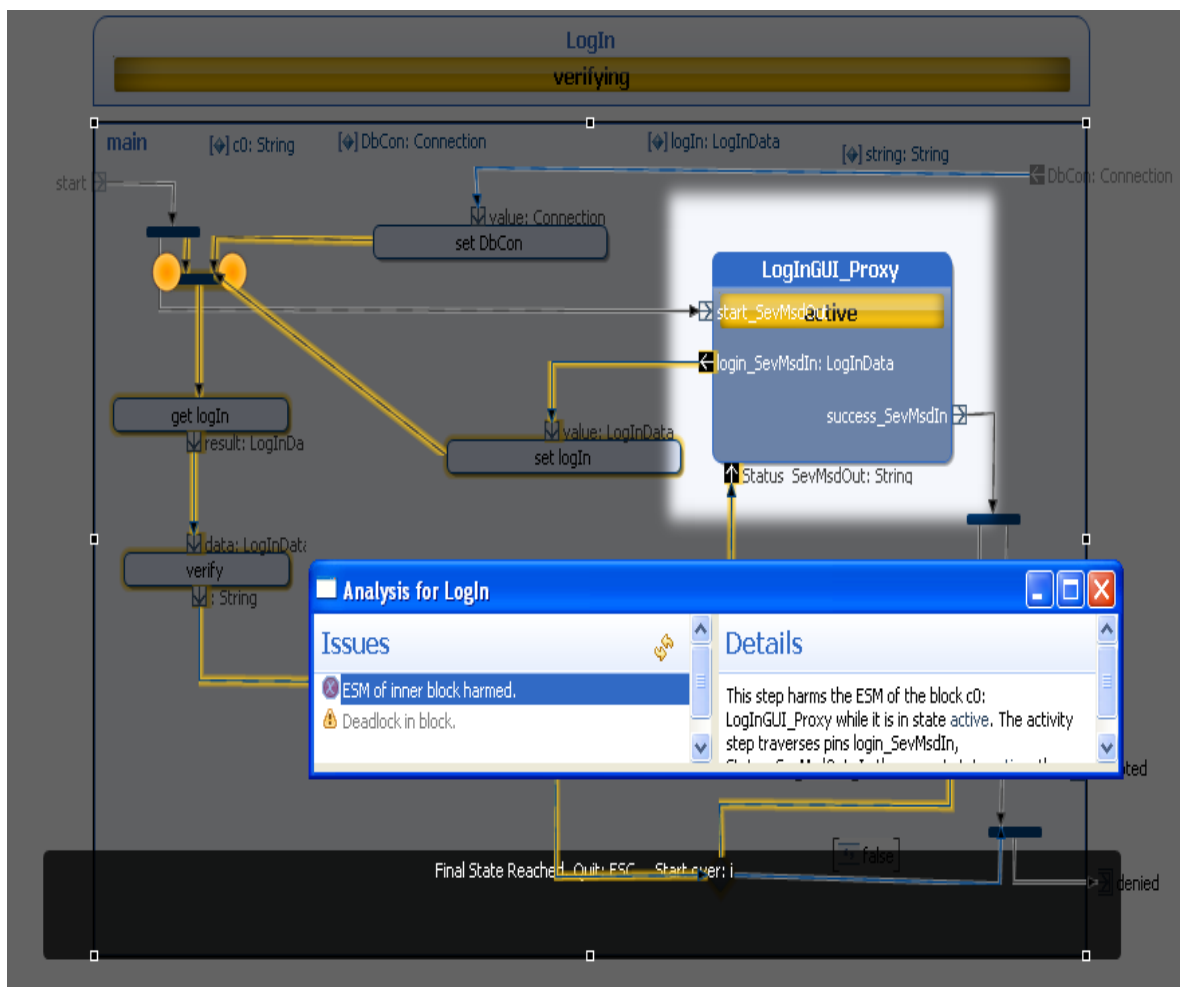


Figure 6.1: Error Situation in the LogIn Activity Block

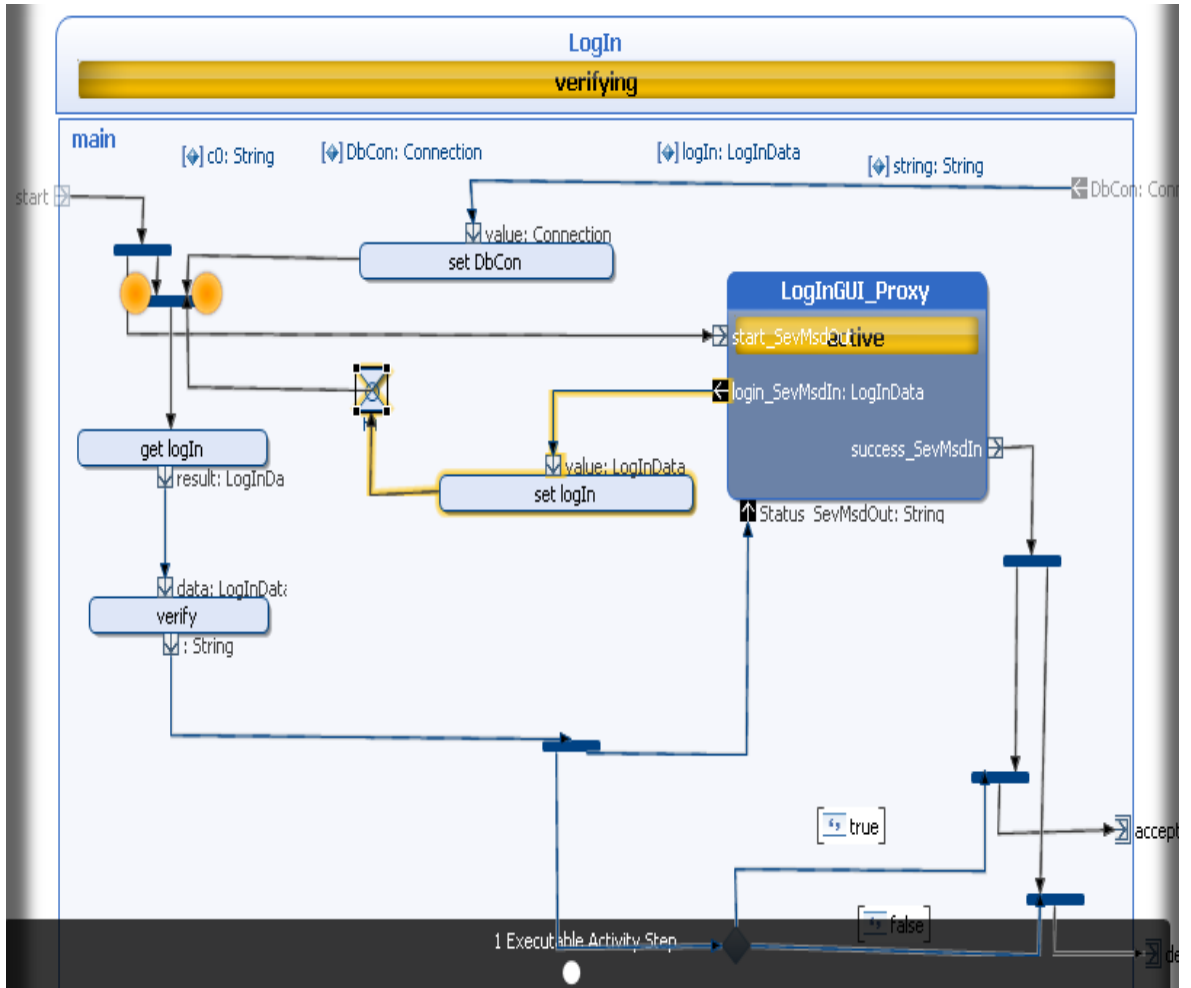


Figure 6.2: Simulation of the LogIn Activity Block

The timer we have put in the LogIn activity block prohibits the token flow to traverse the *login_SevMsdIn* and *Status_SevMsdIn* pin of the LogInGUI_Proxy in one activity step. The simulation in Figure 6.2 shows the situation. With the timer placed in the LogIn activity block, the token flow traverses the *login_SevMsdIn* pin of the LogInGUI_Proxy activity in an activity step and in the following activity step the token flow traverses the *Status_SevMsdIn* pin. The LogInGUI_Proxy goes in a transition from an *active* phase to an *active1* phase in between these two activity steps. This allows the token in the Login activity to traverse the *Status_SevMsdIn* pin, when the LogInGUI_Proxy is in an *active1* phase. So, now the activity of the LogIn block conforms with the ESM of its inner block LogInGUI_Proxy.

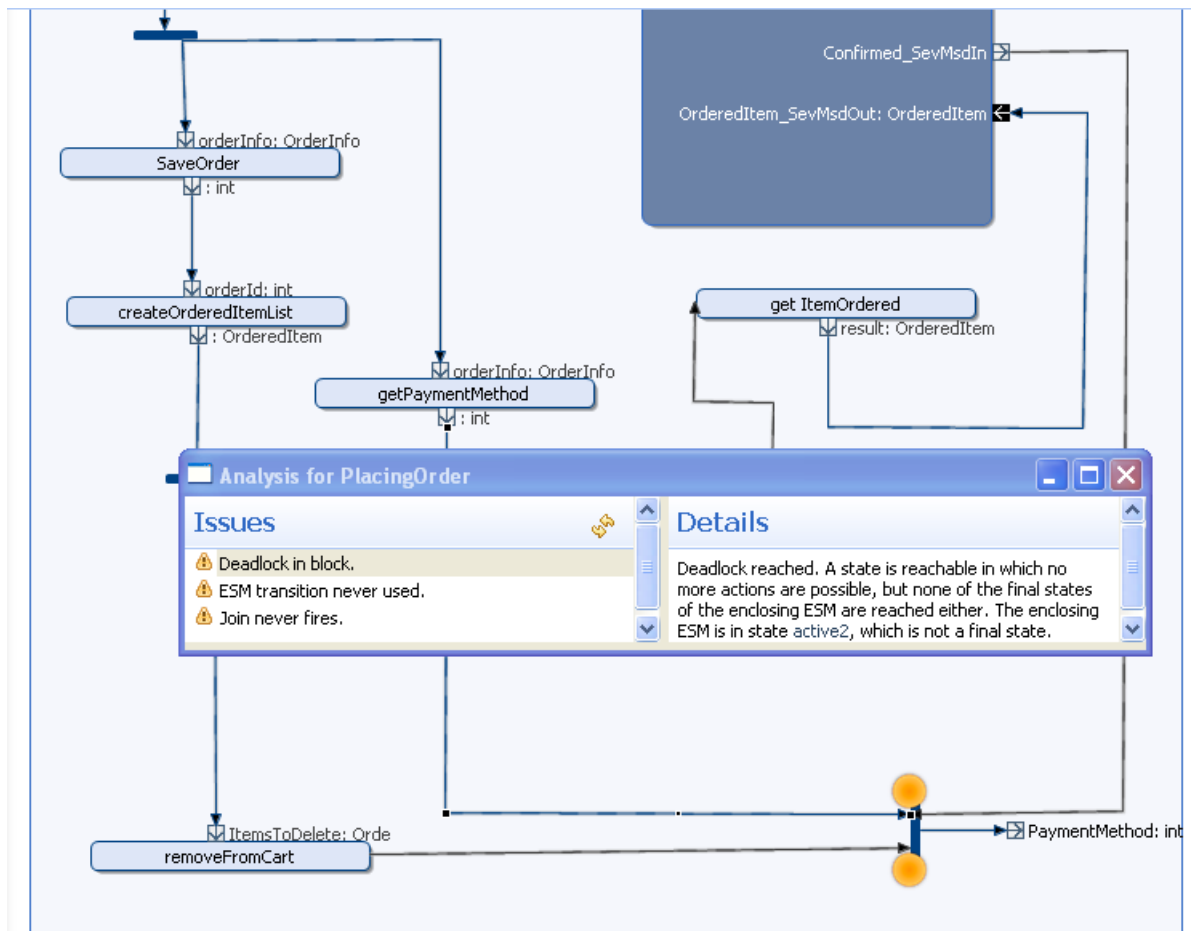


Figure 6.3: Error Situation in the PlacingOrder Activity Block

This removes the error situation. We have not set any time delay in the timer, so there will be no performance problem.

The same error situation described above happens in the `ServiceQuery`, `AddtoCart`, `PlacingOrderEnabler` and in the `PlacingOrder` building blocks. We have removed the error situation in those building blocks in the same way we have described above.

Notice, we have put one more timer with a zero delay in the `PlacingOrder` building block after having set the payment method information in a variable. Without this timer we have been notified with three warnings- deadlock in block, ESM transition never used and join never fires. Figure 6.3 shows this error situation.

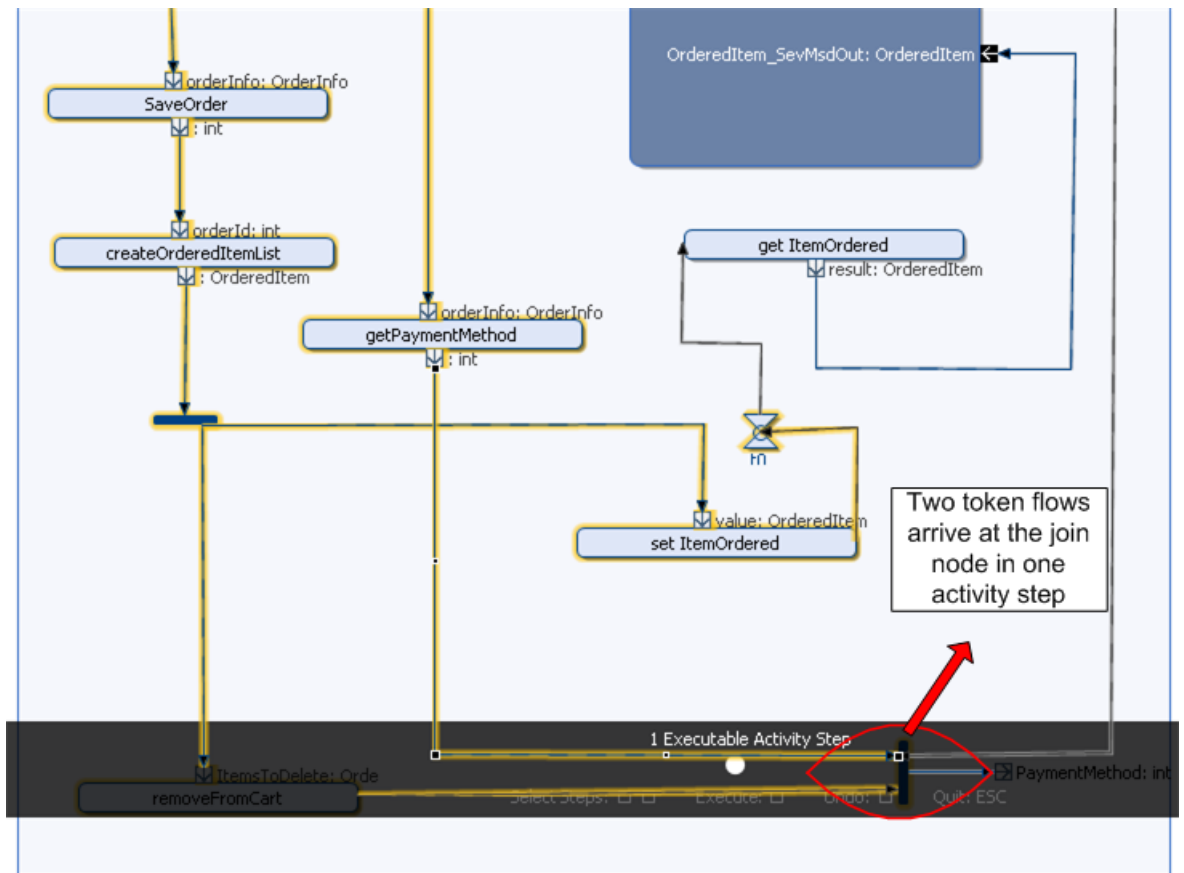


Figure 6.4: Two Token Flows arrive at the Join Node in One Activity Step

Without this timer, two token flows arrive at the join node (placed at just before the terminating output pin *PaymentMethod*) in one activity step. Figure 6.4 shows the corresponding simulation of this activity step. This creates a deadlock situation in the activity of the *PlacingOrder* building block. Putting a timer on the way of one of the token flow allows the two token flows to arrive at the join node in two activity steps. This removes the introduced error situations.

6.1.2 Analysis on the Specified System

During the analysis of our Online Shopping System, we have been notified an error situation as shown in Figure 6.5. The error situation shows that the activity of the system block has harmed the ESM of the *LogIn* building

block. According to the ESM of the LogIn building block shown in Figure 5.8 in section 5.4.1, when the activity of the Login building block is in an *verifying* state, a token flow can traverse either the *accepted* pin or the *denied* pin. But, with the error situation we have been notified, the activity step traverses both the *denied* and *start* pin of the LogIn activity block. This harms the ESM of the LogIn building block.

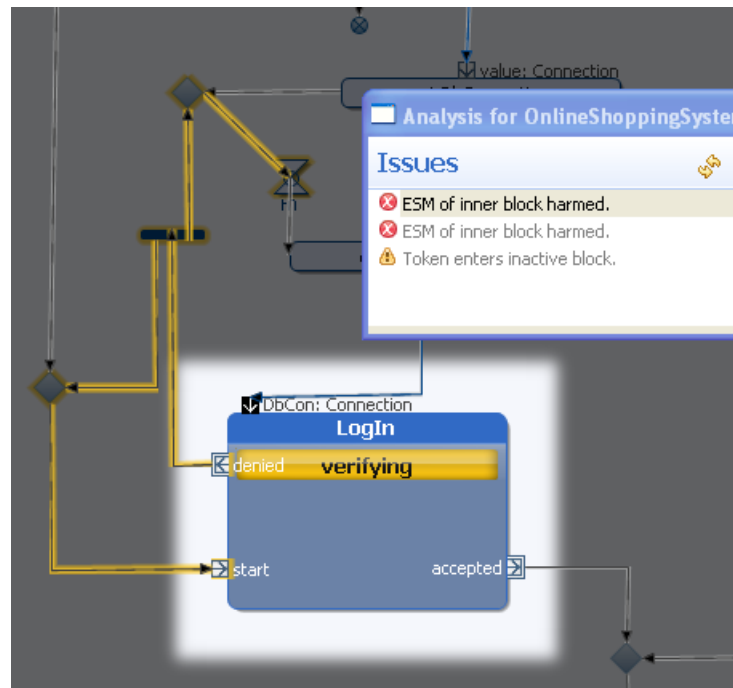


Figure 6.5: Error Situation in the Online Shopping System

To remove this error situation we have put a timer with a zero delay on the way of the token flow that gets out from the *denied* pin of the LogIn block and traverses the *start* pin of the LogIn activity block. Because of this timer, the token flow getting out from the *denied* pin of the LogIn activity block can traverse the *start* pin of the LogIn block in two activity steps. Figure 6.6 shows the simulation of such situation. The timer lets the LogIn activity block to go in a transition from a *verifying* phase to the *final* phase before the token flow traverses the *start* pin of the LogIn building block. This allows the activity of the system to conform with the ESM of the LogIn building block.

The same error situation that we have mentioned above occurs, when the activity step of the system traverses the *Success* and the *Start* pin of the

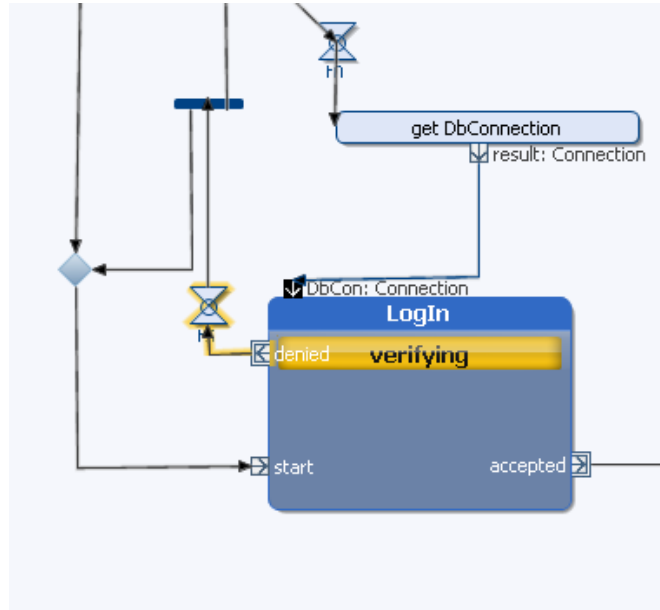


Figure 6.6: Error Situation is Removed Placing a Timer

ServiceQuery block while it is in a $s0$ state. The ESM of the ServiceQuery building block shown in Figure 5.12 in section 5.6.1 says that, while the activity of the ServiceQuery building block is in an $s0$ phase, the token flow can traverse either the *error* pin or the *Success* pin. We have removed the error situation by putting a timer with a zero delay on the way of the token flow that gets out from the *Success* pin of the ServiceQuery block and traverses the *Start* pin of that block.

6.2 Implementation of the Client-Server Communication Model

With our proposed functionalities that we have discussed in section 3.3, the GWT code generator has been implemented at the Telematics department of NTNU. With the implemented GWT code generator, we have implemented our client-server communication model that we have presented in section 3.2.

We show the server system containing the client-server communication model GWTGUITesting_Proxy along with the ESM of the GWTGUITesting_Proxy Building block in Figure 6.7.

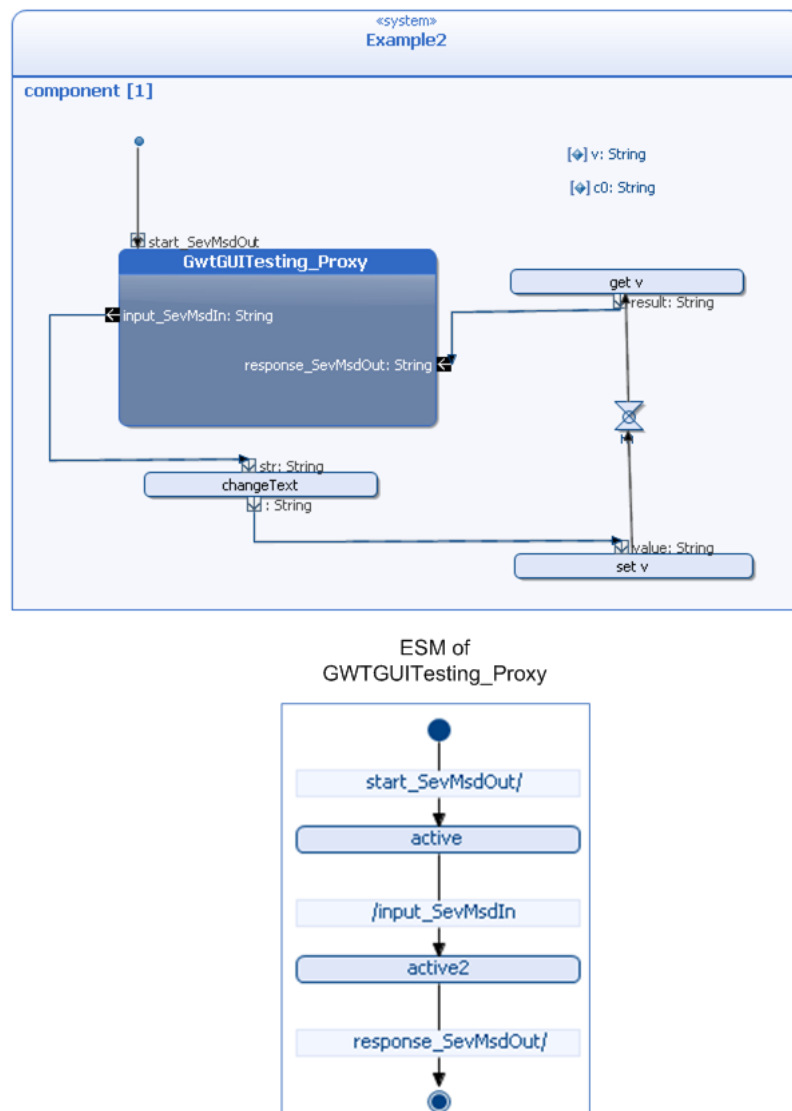


Figure 6.7: Example Server System Containing the Client-Server Communication Model

When we have implemented this server system with the GWT code generator, we have found that, the GWT code generator has created the GWT application project structure as we have proposed. We also have found that, the generated code from our server system model has been placed in a Java Servlet in the generated server side package. Moreover, we have found that, three method signatures with their asynchronous version corresponding to the input/output pins of the GWTGUITesting_Proxy building block have

been automatically generated and placed in the generated client side package. Figure 6.8 shows the automatically generated method signatures.

```
Void start_SevMsdOut()
Void input_SevMsdIn(Java.lang.String text)
Java.lang.String response_SevMsdOut()
```

Figure 6.8: Automatically Generated Methods from the Input/Output Pins of the GWTGUITesting_Proxy Building Block

We have developed the client side programming manually in order to complete the development of the AJAX web application. In the client side program, we have used the methods those have been generated automatically by the GWT code generator. We have called these methods in the client program, in order to get services from the server.

In the client side programming, firstly, we have developed a simple user interface containing a text box and a Send button UI element. In the text box, user can input a text. When the user will click the button the text will be sent to the server and the server's reply will be shown in a pop up panel UI element.

In order to make possible the above mentioned functionality, we have done some client side programming. In the Send button's click action, we have first called the *start_SevMsdOut()* method. In the *onSuccess()* method of the Callback instance of the *start_SevMsdOut()* method, we have called the *input_SevMsdIn(String text)* method. This method passes the user given text to the server. When this method returns successfully, we have called the *String response_SevMsdOut()* method. The server passes the reply as this method's return value. So, when this method returns successfully, we have the reply of the server and we have shown that reply in a pop up UI element. Figure 6.9 shows the result of our successful implementation.

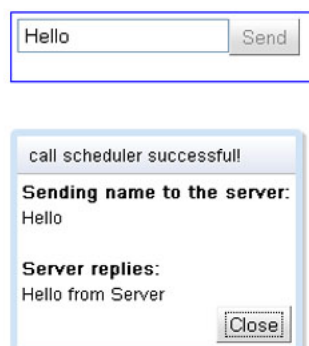


Figure 6.9: Implementation of the Client-Server Communication Model

6.3 Implementation of the ServiceQuery Building Block

We have implemented the ServiceQuery building block putting it in a System block in order to get the available service names in an existing Online Shopping Store that sells services as products. The existing Online Shopping Store, which is the ISIS Project Store [7] is a project carried out under the applied research project *Infrastructure for Integrated Services (ISIS)* in Telenor, Trondheim of Norway. In order to get the available service names from the ISIS Project Store, we have first studied the data storage structure of the Store.

The ISIS Project Store uses a freebXML [5] registry for storing services. FreebXML is an open source implementation of the ebXML repository/registry. In the ebXML repository any type of digital content can be stored as a repository item. The registry manages the metadata describing the repository items. We can explain the ebXML repository/registry as a digital library where the repository is analogous to shelves containing digital contents and the registry is analogous to card catalog having the facility to locate the digital content.

For accessing or using the ebXML registry Java provides an API named JAXR API [6]. Using the JAXR API and providing the registry location address of the ISIS Project Store, we can establish a connection with the ISIS

Project Store. We have put the corresponding code in the *connect* operation action of our ServiceQuery building block. Using a SQL query we can search the information about the digital content in the registry as well as the content itself in the repository. So, we have put a SQL query for searching the names of the stored services in the ISIS Project Store registry. We have put this query in the *getService* operation action of our ServiceQuery building block. When we have implemented the system containing the ServiceQuery

```

Void Start_SevMsdOut()
Void serviceRequest_SevMsdIn(Java.lang.String text)
Java.lang.String getservice_SevMsdOut()
Void success_SevMsdIn()

```

Figure 6.10: Automatically Generated Methods from the Input/Output Pins of the ServiceQuery_Proxy Building Block



Figure 6.11: Implementation of the ServiceQuery Building Block

building block with the GWT code generator, the methods corresponding to

the input/output pin of the ServiceQuery_Proxy have been generated automatically. Figure 6.10 shows the generated methods. Also, the asynchronous version of the generated methods and the server side code has been generated automatically from our system model. We have used these methods in the client side programming to call on the server. Figure 6.11 shows the result of our implementation. When a user sends a request by clicking the send button, the request is sent to the server by calling the *serviceRequest_SevMsdIn()* method. The server replies with the available service names in the ISIS Project Store as a return value of the *getService_SevMsdIn()* method. We have shown the service names in the pop up UI element.

6.4 Implementation of a Test System

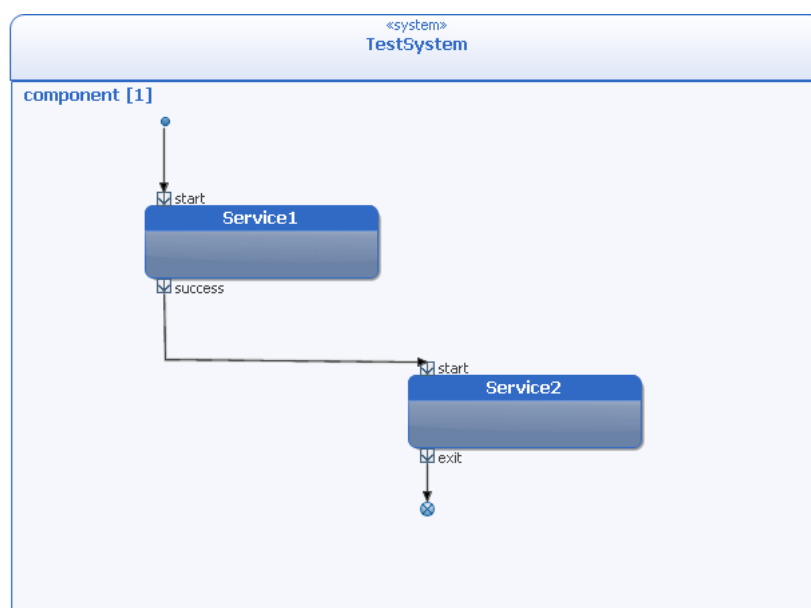


Figure 6.12: Test System Model

We have not been able to implement the whole system that we have modeled because of the time limitation. Rather, we have modeled a simple test system whose activity resembles our specified Online Shopping System. This implementation proves that, it is possible to implement our specified Online Shopping System. In the following sub sections, we describe the test system and the activity blocks we modeled for the test system.

Figure 6.12 shows the test system we have modeled. From the initial node the activity is started and the instance of Service1 building block is activated by the token flow passed via its *start* pin. After providing the service to the client, the Service1 activity is terminated by a token flow passed via its *success* pin. This token flow starts the activity of the Service2 block. Service2 block goes in a termination, when a token flow is passed via its *exit* pin and the token flow is terminated by a flow final node.

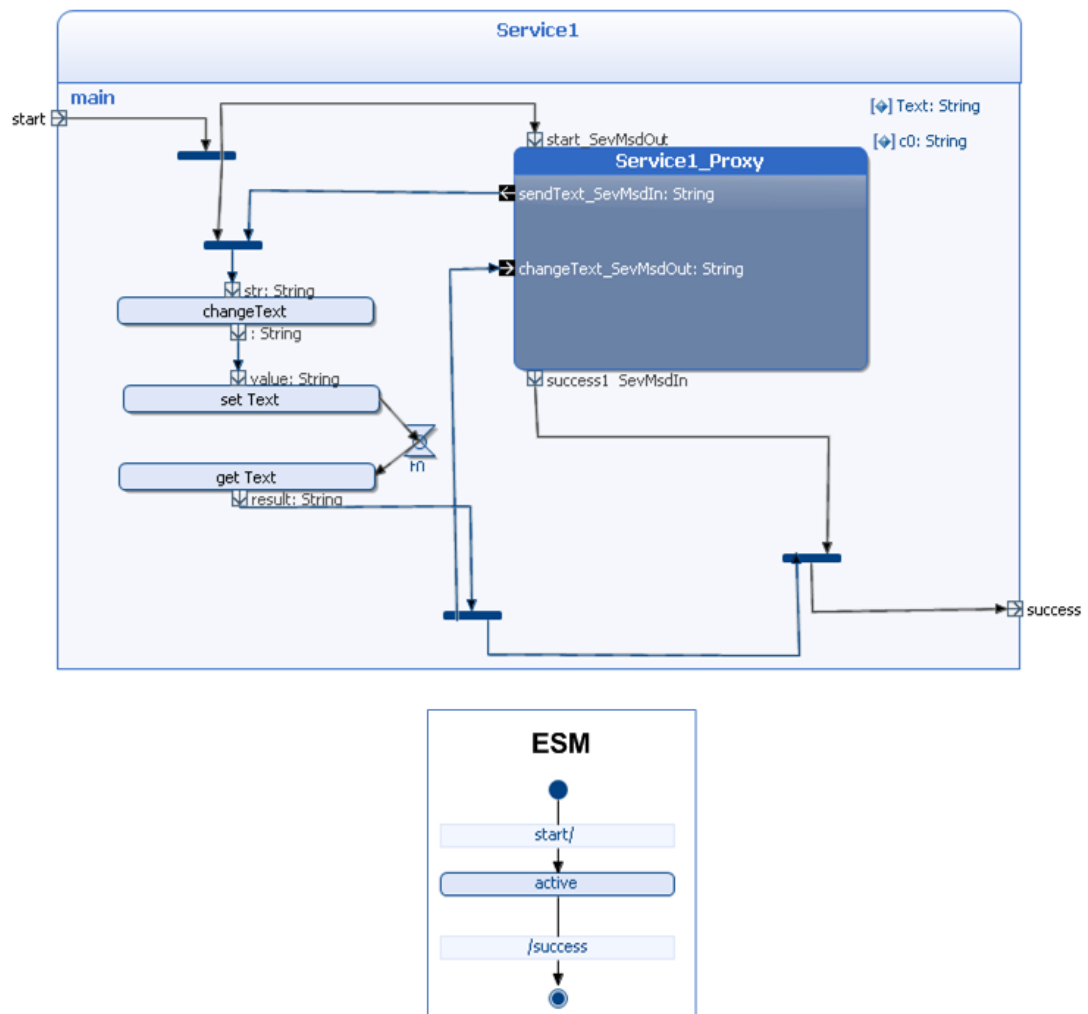


Figure 6.13: Activity and ESM of the Service1 Building Block

Figure 6.13 shows the activity of the Service1 building block along with its ESM. Service1 building block changes the text sent by the user via

the *sendText_SevMsdIn* pin of its inner block *Service1_Proxy* block. Figure 6.14 shows the *Service1_Proxy* activity block with its ESM. *Service1* building block sends the changed text back to the client via the *changeText_SevMsdOut* pin of the *Service1_Proxy* activity block.

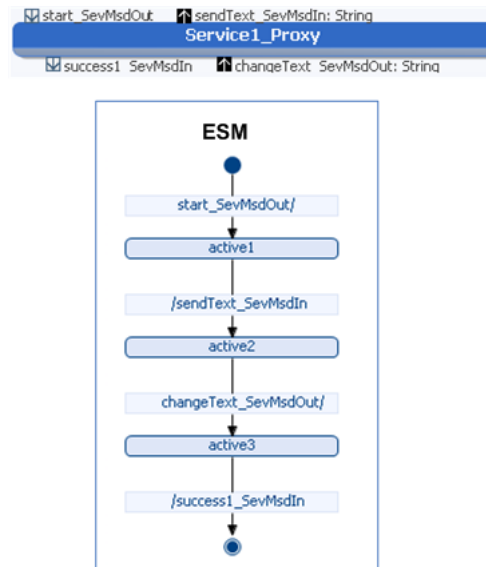


Figure 6.14: Activity and ESM of the *Service1_Proxy* Building Block

Figure 6.15 shows the activity of the *Service2* building block along with its ESM. This building block sends a text to the client upon the request of the user. The request is sent to the activity of this block via the *request_SevMsdIn* output pin of its inner block *Service2_Proxy*. *Service2_Proxy* represents the client behavior. Figure 6.16 shows the activity of the *Service2_Proxy* block and its ESM. The activity of the *Service2* building block sends the reply to the client via the *getService_SevMsdOut* input pin of the *Service2_Proxy* block.

We have done a client side programming using the automatically generated methods from the input/output pins of the *Service1_Proxy* and *Service2_Proxy* building block. When the user inputs a text in the text box and clicks the *service1* button, the text is sent to the server by the *sendText_SevMsdIn()* method and the server sends the reply as a return value of the *sendText_SevMsdIn()* method. Figure 6.17 shows the implementation result.

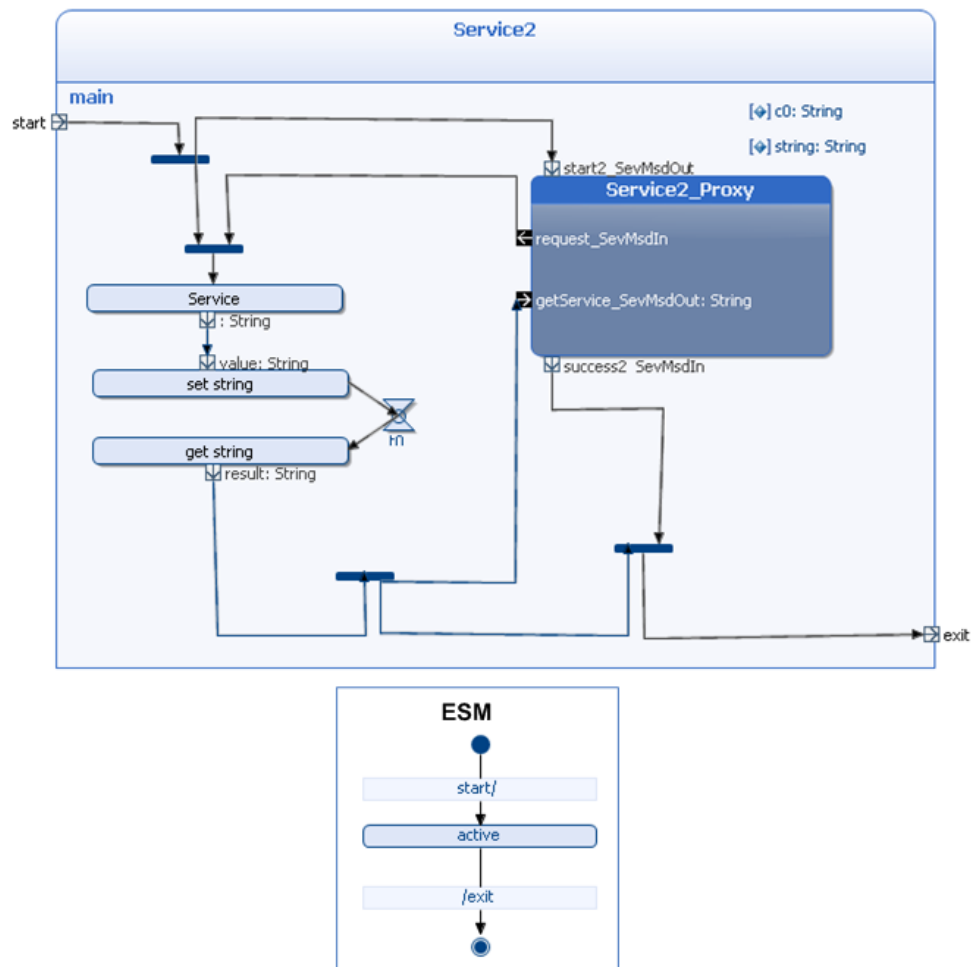


Figure 6.15: Activity and ESM of the Service2 Building Block

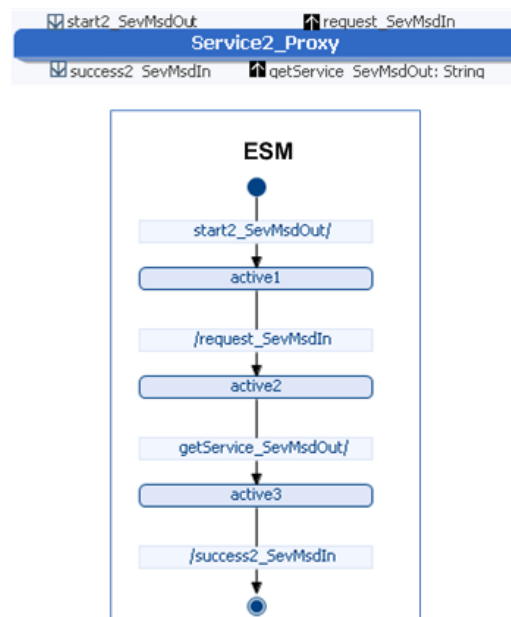


Figure 6.16: Activity and ESM of the Service2_Proxy Building Block

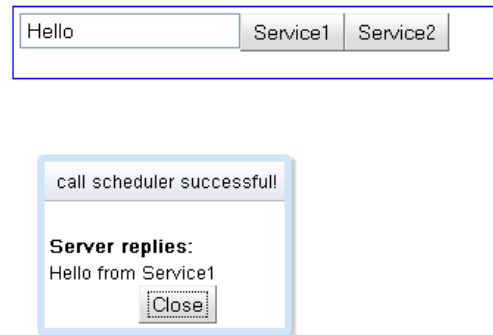


Figure 6.17: Reply from Service1 Building Block

When the reply comes from the server (Service1 block), the service2 button is enabled. When the user clicks the service2 button, a request is sent to the server (Service2 block) with *request_SevMsdIn()*. The server returns a reply as a return value of the *getService_SevMsdOut()* method to the client. The reply is shown in a pop up panel as shown in Figure 6.18.

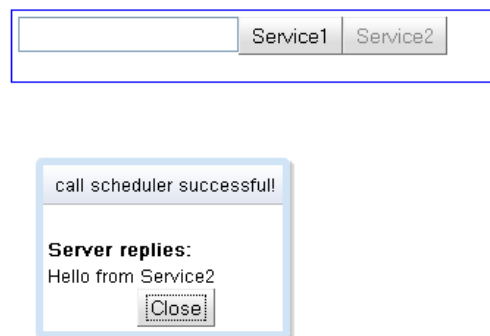
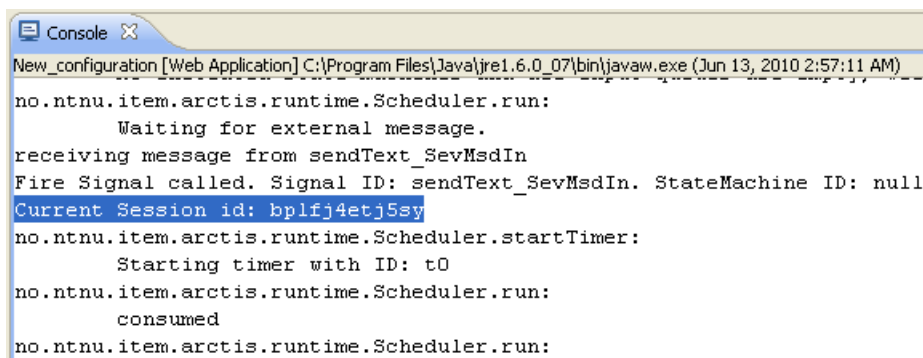


Figure 6.18: Reply from Service2 Building Block

6.5 Multisession Issue

Every web application has the ability to handle multisession. It is a common scenario for web applications that several users may access the same web

application at the same time. So, the server of the web application creates a session for each user who is accessing the web application. The Server performs the session tracking maintaining a unique session id of the user.

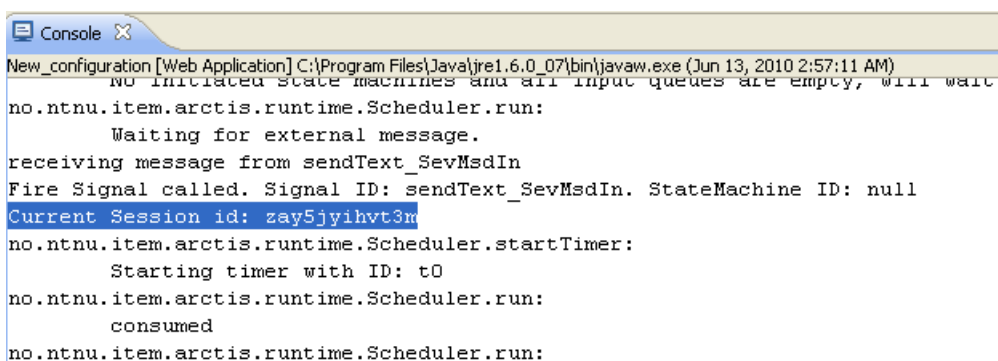


```

New_configuration [Web Application] C:\Program Files\Java\jre1.6.0_07\bin\javaw.exe (Jun 13, 2010 2:57:11 AM)
no.ntnu.item.arctis.runtime.Scheduler.run:
    Waiting for external message.
receiving message from sendText_SevMsdIn
Fire Signal called. Signal ID: sendText_SevMsdIn. StateMachine ID: null
Current Session id: bplfj4etj5sy
no.ntnu.item.arctis.runtime.Scheduler.startTimer:
    Starting timer with ID: t0
no.ntnu.item.arctis.runtime.Scheduler.run:
    consumed
no.ntnu.item.arctis.runtime.Scheduler.run:

```

Figure 6.19: Current Session ID of the Client with the Firefox Browser



```

New_configuration [Web Application] C:\Program Files\Java\jre1.6.0_07\bin\javaw.exe (Jun 13, 2010 2:57:11 AM)
no initiated state machines and all input queues are empty, will wait.
no.ntnu.item.arctis.runtime.Scheduler.run:
    Waiting for external message.
receiving message from sendText_SevMsdIn
Fire Signal called. Signal ID: sendText_SevMsdIn. StateMachine ID: null
Current Session id: zay5jyihvt3m
no.ntnu.item.arctis.runtime.Scheduler.startTimer:
    Starting timer with ID: t0
no.ntnu.item.arctis.runtime.Scheduler.run:
    consumed
no.ntnu.item.arctis.runtime.Scheduler.run:

```

Figure 6.20: Current Session ID of the Client with the Google Chrome Browser

In the server system of the Online Shopping System that we have modeled, we have not reflected the multisession handling activity in the model as described in [42]. However, the implemented web application with our modeled server system can handle multiple session. The code generator with which we have implemented our server system model, places the generated code from our model in a Java Servlet. We have described in section 2.6 that, the application server where the Servlet is loaded handles the session tracking and multiple session handling. So, though we have not reflected the multiple

session handling functionality in our server system model, the application server where it is deployed handles it.

In order to prove what we have described above, we have performed a small test. In our test system model we have put a line of code that gets current session id of the accessed user. We have also put a line of code to print that session id. After implementing the system, we have accessed the web application with two different web browsers (Firefox and Google Chrome). In the console, we have found that for the two different browser clients two different session ids have been created as shown in Figure 6.19 and Figure 6.20. So, it proves that multiple session handling is not a problem for our deployed web application.

Chapter 7

Discussion

The engineering method SPACE and its tool suits (Arctis and Ramses) pave the way of model driven development process by specifying systems as hierarchies of activities encapsulated by ESMs. Reusing self contained building blocks (activities encapsulated with their ESMs) makes the task of specifying systems easier for the developers, as they do not need to look into the internal details of the building blocks. Works of several domains like embedded sensor systems [48], trust management [30], and web service-based telecom services [47] have been done on Arctis. Our work has added another working domain on Arctis platform. We have studied the model driven web application development on Arctis. The achievement from this thesis is twofold: model driven web application development and inclusion of the web application development domain to Arctis platform.

We have integrated the AJAX web application development partially in Arcits as we have been able to model only the server logic of AJAX based web applications. We have done the client side programming manually. In the following sections, we present an evaluation of our resulting work, limitations of our work and future plans of our work.

7.1 Evaluation of the Resulting Work

For modeling the client-server communication we have used Proxy local block of Arctis. In our model the Proxy building block represents the client side behavior. According to our model, the client invokes the service of the server through the input/output pins of the Proxy building block. The input/output pins of the Proxy building blocks ultimately become the function

calls on the server. We have designed this model studying the GWT RPC communication mechanism provided by the GWT framework. With this model, we have also developed and implemented an AJAX web application with GWT framework. However, our model will work for all the AJAX web applications whose client server communication is based on RPC. Only the code generator should be modified according to the requirement of the implementation.

We have specified an Online Shopping System designing some reusable building blocks. We have designed a MySQL building block that provides a connection to a SQL database. This building block can be reused in any applications, where the system needs to be connected with a SQL database. The other building blocks (LogIn, ServiceQuery, AddtoCart, PlacingOrderEnabler and PlacingOrder) we have designed use our client-server communication model. So, their reusability is limited to those AJAX web applications whose client-server communication is based on RPC. We have designed a LogIn building block that authenticates a user verifying his/her user information. Most of the web applications require user authentication services. So, our LogIn building block has a high reusability property.

We have designed the ServiceQuery building block to get service names from an existing Online Shopping Store (the ISIS Project Store). For the implementation purpose, we have designed the exception event notifications specific to the connection exceptions of the ISIS Project Store. So, we cannot say that this building is totally reusable for getting service names or information from any Online Shopping Store. However, the activity of this building block shows in general the modeling of connecting to a third party server and fetching some information from them. So, it helps the developers who want to model such functionalities.

The AddtoCart building block provides the service of adding desired items into one's shopping cart as many times as he wants before checking out. The PlacingOrder building block places an order for the user. These two building blocks contain the core functionalities that an Online Shopping System requires for providing AddtoCart and PlacingOrder services. So these two building blocks can be reused in any Online Shopping System.

We have attempted to implement our specified system model. For the time limitation we have not been able to implement the whole system model. We have implemented first our client-server communication model successfully. Then we have implemented one of our designed building blocks (the ServiceQuery block). Finally, we have implemented a simple test system resembling our specified system model. Our implementation success proves that our

system model is implementable.

We have actually modeled the server side logic of AJAX web applications. We have also proposed the functionalities of a code generator to generate the code from our model and implement an AJAX web application with GWT framework with that server side code. So, in general we can say that, our work can raise the productivity of the AJAX web application development, as the code generator automatically generates server side code from our server model. Also, we think that our server model of Online Shopping System has a high business prospect. Many software vendors develop online shopping softwares and sell them as e-commerce softwares. Many organizations use those online shopping softwares in their web sites. In this perspective, our work can provide some sort of automation in the development of online shopping softwares.

7.2 Limitations

The web application we have developed is done partially in the model driven way. We have modeled only the server side or business logic. Our web application development lacks the client side modeling.

With our model, it is not possible to store information across multiple sessions. For example, a messenger application may store the information of the users online in a runtime memory. Modeling of such applications is not possible with our model. This is because, the generated code from our model is placed in a Java Servlet class. From section 2.6 we know that the Servlet instance is loaded only once in the server. When each client requests for a service in the server, a new thread in the Servlet instance is created for executing our model generated code. Our model is not actually generated as a Servlet class. So, if we store the data about the accessed users in a runtime universal variable in our server system model, the variable can hold only the last user's data as that runtime variable is not treated as a universal variable instance in the Servlet class.

7.3 Future Work

Currently we have done only the modeling of the server side logic of AJAX web applications in Arctis. Client side development also should be done in the model driven way. Modeling client side development requires a web UI

element editor to be integrated in Arctis. Integrating Web UI element editor in Arctis can be a future research work.

For storing information across multiple sessions requires the server system model to be generated as a Servlet class after the implementation. Currently our server system model is treated as a service of a Servlet Class that the Servlet Class provides to the client, as the model generated code is placed in a Servlet class generated by the code generator. To generate a Servlet class itself from our server system model, it requires reflecting of the multiple sessions handling functionality in the model, as a Servlet Class does by creating a new thread for each request. So, modeling multiple sessions handling in our server system model can be a future work.

Bibliography

- [1] WWW page of the article on AJAX: <http://adaptivepath.com/publications/essays/archives/000385.php>.
- [2] WWW page of WebRatio: <http://www.webratio.com>.
- [3] WWW page of Argouml: <http://argouml.tigris.org>.
- [4] WWW page of serializable Javadoc: <http://java.sun.com/j2se/1.3/docs/api/java/io/Serializable.html>.
- [5] FreebXML . WWW page of freebXML: <http://ebxmlrr.sourceforge.net/>.
- [6] Java API for XML Registries (JAXR). WWW page of JAXR API: <http://java.sun.com/webservices/jaxr/index.jsp>.
- [7] The ISIS Project Store . WWW page of ISIS Project Store: <http://shop.isisproject.org:8180/isisStore/>.
- [8] ASADI, M., AND RAMSIN, R. Mda-based methodologies: An analytical survey. 2008, pp. 419–431.
- [9] BAUMEISTER, H., KOCH, N., AND ZHANG, G. Modelling adaptivity with aspects. In *International Conference on Web Engineering (ICWE 2005)*.
- [10] BOCK, C. UML 2 Activity and Action Models. Tech. rep., July-August 2003.
- [11] BOCK, C. UML 2 Activity and Action Models, Part 2: Actions. Tech. rep., September-October 2003.
- [12] BRAM SMEETS, URI BONESS AND ROALD BANKRAS. *Beginning Google Web Toolkit: From Novice to Professional*. APress, September 2008.

- [13] BRÆK, R, HAUGEN, Ø. *Engineering Real Time Systems : An Object-Oriented Methodology Using SDL*. The BCS Practitioner Series. Prentice Hall, 1993.
- [14] CACHERO, C., GÓMEZ, J., AND PÁRRAGA, A. Lecture notes in computer science 1 extending uml for the migration of legacy systems to the web, 2002.
- [15] CERI, S., FRATERNALI, P., AND BONGIO, A. Web modeling language (webml): a modeling language for designing web sites. pp. 137–157.
- [16] CERI S., F. P. M. M. Conceptual modeling of data-intensive web applications. *Internet Computing, IEEE 6* (2002), 20 – 30.
- [17] CHRIS ULLMAN AND LUCINDA DYKES. *Beginning Ajax*. Wiley Publishing, Inc, 2007.
- [18] CONSORTIUM, U. Ubiquitous web applications. In *In Proceedings of the eBusiness and eWork Conference* (October 2002).
- [19] ESCALONA, M. J., AND KOCH, N. Metamodeling the requirements of web systems.
- [20] FRANK ALEXANDER KRAEMER AND PETER HERRMANN. Automated Encapsulation of UML Activities for Incremental Development and Verification. In *Proceedings of the 12th Int. Conference on Model Driven Engineering, Languages and Systems (Models), Denver, Colorado, USA, October 4-9, 2009* (2009), Andy Schürr and Bran Selic, Ed., vol. 5795 of *Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, pp. 571–585.
- [21] GARZOTTO, F., PAOLINI, P., AND SCHWABE, D. Hdm - a model-based approach to hypertext application design. *ACM Trans. Inf. Syst.* 11, 1 (1993), 1–26.
- [22] GERBER, A., MICHAEL LAWLEY, KERRY RAYMOND, J. S., AND WOOD, A. Transformation: The missing link of mda. In *In: Proceedings of the 1st International Conference on Graph Transformation* (2002), Springer, pp. 90–105.
- [23] GÓMEZ, J., AND CACHERO, C. Oo-h method: extending uml to model web interfaces. 144–173.
- [24] GÓMEZ, J., AND CACHERO, C. Model-driven web development with visualwade. 144–173.

- [25] GOMEZ, J., CACHERO, C., PASTOR, O., AND SPAIN, V. Extending a conceptual modelling approach to web application design. In *In 12 th International Conference on Advanced Information Systems (CAiSEŠ00*.
- [26] GROUP, O. M. Meta Object Facility (MOF) Core Specification Version 2.0. Tech. rep., January 2006.
- [27] GROUP, O. M. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Tech. rep., July 2007.
- [28] GROUP, O. M. Unified Modeling Language: Superstructure, version 2.1.2. Tech. rep., November 2007.
- [29] HENNICKER, R., AND KOCH, N. A uml-based methodology for hypermedia design. In *Proc. of UML 2000 Conference* (2001), Springer Verlag, pp. 410–424.
- [30] HERRMANN, P., AND KRAEMER, F. A. Design of Trusted Systems with Reusable Collaboration Models. In *Trust Management* (2007), S. Etalle and S. Marsh, Eds., vol. 238, IFIP International Federation for Information Processing, Springer, pp. 317–332.
- [31] HERRMANN, P., AND KRUMM, H. A framework for modeling transfer protocols. *Computer Networks* 34, 2 (2000), 317 – 337.
- [32] HOUBEN, F. F. G. J., AND VDOVJAK, R. An rmm-methodology for hypermedia presentation design. In *IN PROC. ADVANCES IN DATABASES AND INFORMATION SYSTMES, LNCS 2151* (2001), Springer, pp. 323–337.
- [33] JAKOB FRYDENBERG. *Generation of Web applications from UWE models*. Bachelor of Engineering, IT Thesis, Technical University of Denmark, May 2008.
- [34] JAMES RUMBAUGH, IVAR JACOBSON, AND GRADY BOOCH. *The Unified Modeling Language*, second edition ed. Addison-Wesley, 2005.
- [35] JASON HUNTER AND WILLIAM CRAWFORD. *Java Servlet Programming*, second edition ed. O’Reilly Media, 2001.
- [36] KNAPP A., KOCH N., M. F., AND G., Z. Argouwe: A case tool for web applications. In *In Proc. of 1st Int. Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE03)* (September 2003).

- [37] KOCH, N., AND KRAUS, A. Towards a common metamodel for the development of web applications. In *Proc. of the 3 rd Int. Conf. on Web Engineering (ICWE)*.
- [38] KOCH, N., AND KRAUS, A. The expressive power of uml-based web engineering, 2002.
- [39] KRAEMER, F. A. Profile for Service Engineering: Executable State Machines. AVANTEL Technical Report (2/2006), NTNU , Department of Telematics, NTNU.
- [40] KRAEMER, F. A. UML Profile and Semantics for Service Specifications. AVANTEL Technical Report (1/2007) NTNU, Department of Telematics, NTNU.
- [41] KRAEMER, F. A. Arctis and Ramses: Tool Suites for Rapid Service Engineering. In *Proceedings of NIK 2007 (Norsk informatikkonferanse), Oslo, Norway* (November 2007), Tapir Akademisk Forlag.
- [42] KRAEMER, F. A., BRÆK, R., AND HERRMANN, P. Synthesizing Components with Sessions from Collaboration-Oriented Service Specifications. In *SDL 2007* (September 2007), E. Gaudin, E. Najm, and R. Reed, Eds., vol. 4745 of *Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, pp. 166–185.
- [43] KRAEMER, F. A., BRÆK, R., AND HERRMANN, P. Compositional Service Engineering with Arctis. *Teletronikk 105*, 2009.1 (2009).
- [44] KRAEMER, F. A., AND HERRMANN, P. Formalizing Collaboration-Oriented Service Specifications using Temporal Logic. In *Networking and Electronic Commerce Research Conference 2007 (NAEC 2007)* (USA, October 2007), ATSSMA Inc., pp. 194–220.
- [45] KRAEMER, F. A., AND HERRMANN, P. Transforming Collaborative Service Specifications into Efficiently Executable State Machines. In *Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)* (2007), K. Ehring and H. Giese, Eds., vol. 7 of *Electronic Communications of the EASST*, EASST.
- [46] KRAEMER, F. A., HERRMANN, P., AND BRÆK, R. Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In *Proceedings of the 8th International Symposium on Distributed Objects and Applications (DOA), 2006, Montpellier, France* (2006),

- R. Meersmann and Z. Tari, Eds., vol. 4276 of *Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, pp. 1613–1632.
- [47] KRAEMER, F. A., SAMSET, H., AND BRÆK, R. An Automated Method for Web Service Orchestration based on Reusable Building Blocks. In *Proceedings of the 7th International IEEE Conference on Web Services (ICWS)* (July 2009), IEEE Computer Society, pp. 262–270.
- [48] KRAEMER, F. A., SLÅTTEN, V., AND HERRMANN, P. Model-Driven Construction of Embedded Applications based on Reusable Building Blocks – An Example. In *SDL 2009* (2009), A. Bilgic, R. Gotzhein, and R. Reed, Eds., vol. 5719 of *Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, pp. 1–18.
- [49] KRAEMER, F. A. *Engineering Reactive Systems. A Compositional and Model-Driven Method Based on Collaborative Building Blocks*. PhD Thesis, Norwegian University of Science and Technology, August 2008.
- [50] KRAUS, A., AND KOCH, N. Generation of web applications from uml models using an xml publishing framework, 2002.
- [51] L., B., AND L., M. Beyond Modeling Notations: Consistency and Adaptability of W2000 Models. In *In Proc. of SAC'05, ACM Symposium on Applied Computing* (Santa Fe, USA, 2005), Electronic Communications of the EASST.
- [52] LAMPORT, L. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.* 16, 3 (1994), 872–923.
- [53] LAMPORT, L. *Specifying Systems*. Addison-Wesley, 2002.
- [54] LUCIANO BARESI, SEBASTIANO COLAZZO, L. M., AND MORASCA, S. W2000: A modelling notation for complex web applications.
- [55] MELIÁ, S., AND GÓMEZ, J. The websa approach: Applying model driven engineering to web applications. *J. Web Eng.* 5, 2 (2006), 121–149.
- [56] NORA KOCH, G. Z., AND ESCALONA, M. J. Model transformations from requirements to web system design. In *ICWE '06: Proceedings of the 6th international conference on Web engineering* (2006), Springer Verlag, pp. 281–288.

- [57] NORA KOCH, ALEXANDER KNAPP, GEFEI ZHANG, AND HUBERT BAUMEISTER. Uml-based web engineering.
- [58] (OMG), O. M. G. UML 2 Object Constraint Language (OCL). Tech. rep.
- [59] PROF. MARIO A. BOCHICCHIO, E. A. L. Integrating web systems design and business process modeling. In *ICWE 2005: Proceedings of the 5th international conference on Web engineering (2005)*, Springer, pp. 60–69.
- [60] R. VIDGEN, D. AVISON, B. WOOD AND T. WOOD-HARPER. *Developing Web Information Systems*. Elsevier Science, 2002.
- [61] ROBERT HANSON AND ADAM TACY. *GWT in Action*. Manning, June 2007.
- [62] RYAN DEWSBURY. *Google Web Toolkit Applications*. Prentice Hall, 2008.
- [63] SCHWABE, D., AND ROSSI, G. Developing hypermedia applications using oohdm. In *In Proceedings of Workshop on Hypermedia Development Process, Methods and Models, Hypertextt98 (1998)*, ACM.
- [64] SELIC, B. The pragmatics of model-driven development. *IEEE Softw.* 20, 5 (2003), 19–25.
- [65] SELIC, B. Model-driven development: Its essence and opportunities. *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on O (2006)*, 313–319.
- [66] SEMIA SONIA SELMI, N. K., AND GHEZALA, H. B. Toward a comprehension view of web engineering. *LECTURE NOTES IN COMPUTER SCIENCE (2005)*, 19–29.
- [67] VALDERAS P., F. J., AND V., P. From web requirements to navigational design - a transformational approach. *LECTURE NOTES IN COMPUTER SCIENCE*, 3579 (2005), 506–511.
- [68] YU, Y., MANOLIOS, P., AND LAMPORT, L. Model checking tla+ specifications. In *CHARME '99: Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (London, UK, 1999)*, Springer-Verlag, pp. 54–66.

- [69] ZHANG, G., AND MAXIMILIANS-UNIVERSITÄT MÜNCHEN, L. Model transformations for integrating and validating web application models. In *In Proc. Modellierung* (2006).