

A Domain Specific Language For Model-Based Development of Interactive Web Applications

Dainis Nikitins

Kongens Lyngby 2012

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Declaration Of Authorship

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Author

Date

Summary

In the beginning of a Web application development process it is not possible to fully specify the application entire requirements. Requirements continuously evolve, especially after the Web application emerge and is utilized. Also, in comparison to conventional software, Web applications are exposed to anonymous users with different requirements and backgrounds. This makes definition of the Web application requirements one of the most complicated tasks in the development process.

The objective of the thesis is to propose a solution to model requirements of interactive Web applications by using specific modeling notation to gather, specify and validate user requirements, as well as, by applying model transformations, to generate the code of the application. The hypothesis of the thesis is that usage of graphical user interface (GUI) models together with the models of the processes that are invoked by the user interaction on the GUI improve the requirement specification process and reduce the amount of changes initiated by the customer in the later development phases.

Acknowledgements

First of all, I want to thank my supervisor Ekkart Kindler for his help and assistance throughout the whole project. I would also like to thank Rasmus Skovmark from Edora A/S for valuable input for my project and providing me with the example application.

Furthermore, I would like to thank my parents, who always supported me during my whole studies. Special thanks go to my girlfriend Julia, who supported me during the thesis development process and never doubted that I was going to make it.

Finally, I would like to thank my colleagues and friends that I met during my university life for their great teamwork and collaboration.

Contents

Declaration Of Authorship	i
Summary	ii
Acknowledgements	iii
1 Introduction	1
1.1 Background and focus	4
1.2 Objectives	6
1.3 Example	7
1.4 Thesis structure	8
2 State of the Art	10
2.1 Rich Internet Applications	10
2.2 Model-Driven Architecture	15
2.3 Study of Web engineering methods	35
3 Modeling Notation	42
3.1 Overview	42
3.2 Desing of the Domain Specific Language	43
3.3 Domain Specific Language Metamodel	45
3.4 DSL Metamodel for PIM	48
3.5 DSL Metamodel for PSM	54
3.6 Domain Specific Language Concrete syntax	55
3.7 CIM	55
3.8 PIM	58
3.9 PSM	59
3.10 Summary	60

4	Prototype Implementation	61
4.1	Overview	61
4.2	DSL Editors Implementation	62
4.3	DSL Implementation in EMF	63
4.4	GMF Editors	63
4.5	DSL Implementation in GMF	64
4.6	Models Validation	65
4.7	Developing Transformation	67
4.8	Limitations	68
4.9	Summary	69
5	Conclusions	70
5.1	Thesis summary	70
5.2	Future work	72
A	Appendix A	73
B	Appendix B	79
B.1	Installation	79
B.2	User Guide	80

Introduction

It has been 20 years since Tim Berners-Lee invented the World Wide Web (WWW), a system of interlinked hypertext documents¹ that can be accessed via the Internet. The Web has grown exponentially during these years. Today amount of pages on the Web has increased to 50 billion² [1] and number of Internet users to over 2 billion [2] people.

The Web pages have evolved from simple, static-content pages to more extensive and dynamic applications which have a complexity that is comparable to conventional software systems. Examples of such sophisticated Web systems are so-called Rich Internet Applications (RIAs), which became very popular with the emergence of Web 2.0. The main characteristics of these applications are enhanced user interface behavior and improved interactivity which is achieved through distribution of page computation, data distribution and asynchronous communication realization between client and server [3].

Currently, engineering of RIAs is a new field in the area of Software Engineering. There is a noticeable interest in this topic from the researchers in the Web Engineering discipline and, especially, in the Model-Driven Web Engineering (MDWE). To date there has been developed a number of model-based methodologies for design and development of traditional Web applications. Amongst

¹Hypertext document is a synonym for Web page

²Billion is equated to a thousand million or 10^9

the most prominent are UWE [8], WebML [9], OOWS [10], OOHDM [15] and OOH [11]. These methodologies use different models to address different concerns (content, navigation, presentation, business logic etc.) of a Web application. Furthermore, the application's Web pages and most of the logic can be automatically generated based on these models [12]. However, none of these methodologies is fully suitable for RIAs. The differences between the traditional Web applications and RIAs is that latter must not only respond to navigation through hypertext, but also have to incorporate different presentation behaviors, data and processing distribution, scheduled events, exceptions and more. At the same time, the main focus of the model-based development Web methodologies is on the hypertext structure and navigation of the application and not on usability and interactivity of their user interfaces or client-server communication [3, 4].

Another insufficiency of current Web modeling approaches is that they concentrate on design workflow in the life cycle rather than on understanding requirements of the application. Several surveys and comparative studies conclude that such activities as requirements engineering, tests and quality management are treated with less importance or not included at all in most of the Web methodologies [5, 13, 16]. Many of them use requirement specification methods that stem from traditional software system development such as Unified Modeling Language (UML) context diagrams [17], use cases [8], activity diagrams [8] etc. Some of these methodologies define their own notations [14]. However, the weakness of these methods is the assumption that the customer understands his requirements and can formulate his needs. In most Web applications it is not possible to fully specify their entire requirements in the beginning because they continuously evolve, especially after they emerge and are utilized. One may argue that this is also true for the conventional software systems. However, main difference between Web-based and conventional software system is that latter will be utilized within an organization to complete some specific tasks while Web applications will be exposed to anonymous users with different requirements and backgrounds from outside world. This makes definition of the requirements more complicated [5, 6].

The main objective of this thesis is to design a domain-specific language for model-based development of interactive Web applications which would tackle the aforementioned issues of the traditional Web modeling approaches. One of the focuses of the language will be on the requirement engineering of RIAs. The process of requirements engineering consists of three main activities [13]:

- requirements elicitation,
- requirements specification,

- requirements validation.

There exist several requirement elicitation and specification techniques such as interviewing, use case modeling, prototyping, concept mapping etc. [13]. According to research made by Barry et al. [7] prototyping and storyboarding is one of the most widely used approaches for application development among multimedia and Web industry practitioners. The popularity may be explained by their ability to specify the presentation, navigation and temporal aspects of Web applications [7]. In spite of this, none of the current model-based Web methodologies include this technique as means of requirement elicitation or specification [13]. Lowe et al. in [5] hypothesize that iterative process based on client-developer joint study of partial designs improve clients' understanding and clarify systems' requirement specification. Based on these findings the proposed idea is to create a graphical user interface (GUI) design model which could be used for elicitation and specification of Web application requirements. Basically, using this model, business consultant together with the customer will be able to discuss and capture the requirements by building abstract GUI mock-ups. On the later stages of the development these models could be enhanced by style and layout specifications.

After the requirements have been captured and specified they have to be validated. As stated in [13] it is important step since "through requirement validation the requirements specification is checked to correspond to the user's needs and the customer's requirements." As mentioned earlier RIAs is more complex software than Web 1.0 applications in regards to events that might occur while interacting with the application. The possible set of events includes user interaction on the interface, server internal or temporal events (timeouts, data updates etc.), Web service invocations and so forth. It means that to be able to validate the client requirements there should be a way of capturing these events. Our idea is to create a process model which would represent the process to be executed. The process model will be used to specify what event triggers the process and what activities should be fulfilled after it is triggered. This process model could be then associated with any GUI control. The purpose of this model is twofold: on one hand, it can be used to validate client's requirements by allowing simulating the behavior of the application and, on the other hand, it can be used as an initial input for implementation of a complete behavior of the application. The general idea is that from these requirement artifacts, eventually, it should be possible to derive a domain model of the application or at least a part of it.

The technology we are going to use to develop the solution will be a Model-Driven Development (MDD) approach. In this software development approach models are the primary development artifacts and code is automatically gener-

ated by applying model transformations.

1.1 Background and focus

This project is developed in collaboration with Edora A/S company. The company is specialized in Enterprise Content Management (ECM), but also develops different kind of Web-based software systems. The company maintains an iterative and incremental development process and use Scrum as its primary project management form. In order to support the development process the company has implemented its own in-house code generator. The generator is used in the design phase of a project after a domain model of an application is build. The domain model is used as an input for the code generator which produces data model and part of the business logic of the application.

According to the companies' project manager the most challenging part in the development process is the specification of a domain model. He pointed out that several times he experienced difficulties in capturing customer requirements at the commencement of the project. This was caused by the fact that customers had difficulties in understanding UML use case diagrams and class diagrams which were used to document the project. At the same time they received immediate feedback when they used design prototypes.

This confirms the findings of Lowe et.al [5] that show that design artifacts have an important role in the Web project development process as they help to clarify the system requirements. Lowe et al. [5] argue that difference between Web and traditional software systems is that in the latter the project moves more clearly from the requirements specification to software design and later to the implementation of the system. In Web development, however, the boundaries between requirement specification and the design are blurred, since requirements emerge from the design, rather than preceding and driving the design. This makes definition of the requirements more complicated and use of prototypes become more valuable. As mentioned earlier, the requirement gathering process is also influenced by number of stakeholders involved and by evolutionary nature of Web applications. In Web-based applications the possibility of change is much higher than it is in traditional software applications. This makes design prototypes a good candidate to be used in the requirement gathering process because they can easily be modified to resemble the changes.

However, to date, none of Web modeling methodologies uses the GUI design prototyping as a part of its requirement elicitation or specification process. Nevertheless, there exist a number of different commercial and open source tools

for creating Web application mockups such as Axure, Balsamique, Pencil etc. Some of these tools are equipped with code generators, which are able to produce HTML output based on GUI sketches. However, the disadvantage of these tools is that they provide limited functionality and can at most be used to create click-through prototypes. These prototypes are usually used to capture initial customer requirements and are discarded before the development starts. It would be useful if these prototypes could become a part of overall solution and drive the application development which could be achieved using Model-Driven Development approach.

Concerning RIAs it is not enough with only being able to design the GUI view. Comparing to the Web 1.0 applications one of the main distinguishing characteristics of RIAs is that it employs event-driven interaction. Wright et al. [4] as the most serious issue in the development of the RIAs using Web modeling approaches find the lack or very poor support of event modeling. Therefore, there should be a way to capture these events. The proposed solution is definition of the process model. For instance, if user clicks Login button, the corresponding Login process is initiated which might consist of activities such as validate user input and check user existence in the system. This model would allow defining high level design of application tasks.

In order to develop the modeling language we decided to use a Model-Driven Development approach. The use of this technology allows modeling of different concerns of a Web application. For example, the GUI design model would be used to create and abstract GUI of the application, while process model would define the processes triggered by user interacting with GUI controls. With the use of Model-Driven Development, the main advantage would be that created GUI and process models would become a part of overall solution and allow reuse of the models in further application development phases. In the requirements gathering phase these models provide a possibility to verify the requirements by simulating the application behavior. Thus, making it more understandable to customer and helping capturing all the requirements in an early analysis phase before real programming is started.

Another advantage of MDD is separation of the model from the code. This allows business consultant to concentrate on capturing the requirements of the system, abstracting from the any implementation technology. Later the developer will transform the requirements into platform independent model and then select the specific target platform and automatically generate the code for it. This is very beneficial since there are different languages and frameworks for the implementation of Web-based software. In more details the MDD approach is described in Section 2.2.

1.2 Objectives

The main objective of the thesis is to propose a solution to model requirements of interactive Web applications by using a domain-specific language to gather, specify and validate user requirements, as well as, applying model transformations generate code of the application. This domain-specific language would help in solving the problem of requirement elicitation in a Web system, as well as, allow validating the consistency of requirements by simulating its behavior.

The idea is to develop a domain-specific language based on the Model-Driven Development approach. The overview of the modeling notation is given in Figure 1.1.

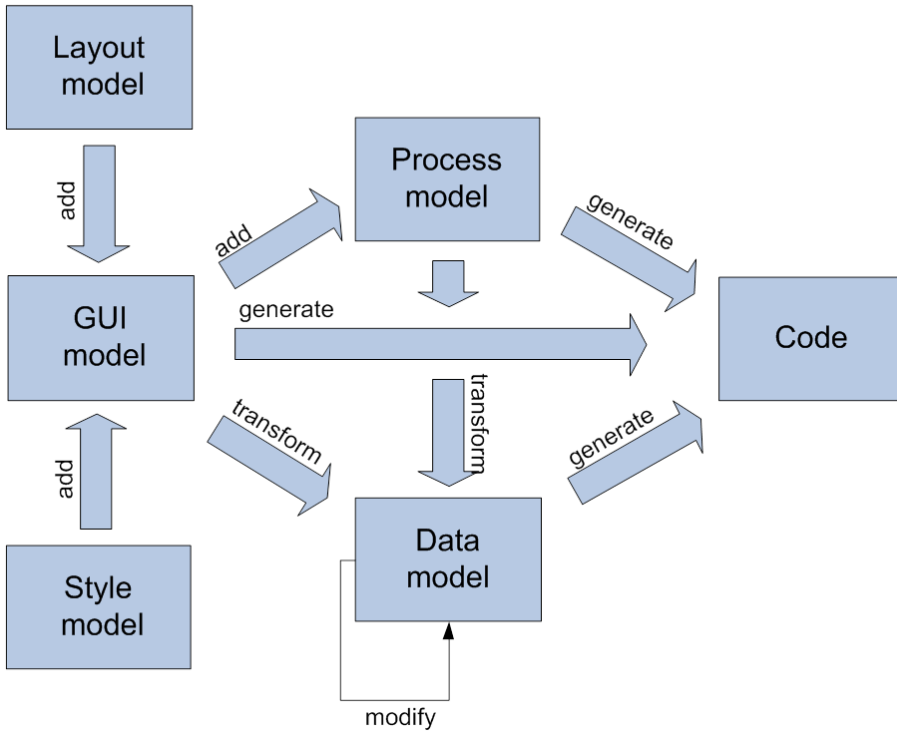


Figure 1.1: Overview of modeling notation and their relation.

In this modeling notation the GUI model is used to specify an abstract presentation of the Web page, the process model is used to capture the different interactions, the layout model is used to specify the layout of the GUI controls

and the style model is used to enhance "look-and-feel" of the GUI. These models can be created separately of each other by different domain experts. For instance, GUI draft can be created by a business consultant, process models can be created by a developer and layout and style models can be defined by a Web designer. The GUI model combined with a process model can be used as an input to deriving a data model of the application. In more detail the design of the modeling language is discussed in the Chapter 3.

In order to create the GUI model the Web 2.0 applications that use RIA technologies will be analyzed. Also since RIAs user interfaces are extended by using widgets that are common in desktop environments, we will look into user experience interaction guidelines of different desktop software vendors such as Windows, Apple etc. To specify the process model, the common reactions to the events in RIA will be examined. The style and layout models will be based on the best practices in design of Web interfaces.

As a proof of concept a prototype tool implementing the GUI design model, process model, style model and layout model will be created. The software will include editors for editing the models, possibility to adjust some model transformations, as well as the code generation. To test the generated application it will be possible to deploy it on a Web server. Using the implemented prototype tool it will be possible to develop and generate operational Web application from its models. In more details it is described in Chapter 4.

In the next section we present the Conveyor operator application provided by the Edora A/S company. We will use this application to evaluate our modeling language.

1.3 Example

As a an example we will use an existing business application developed by Edora A/S. This application is used to control a manufacturing process at a factory. The application is a Web-based application built on the Microsoft .NET platform and written in ASP.NET and C#. It consists of 4 Web pages. The architecture and the navigation possibilities between the Web pages of the application are shown in the Figure 1.2.

To start using the application the conveyor operator has to login into the system using his credentials. If the login is successful, the application redirects user from the login page to the main page. Otherwise, if login fails or any other error occurs while using the system, the conveyor operator is notified with an appropriate

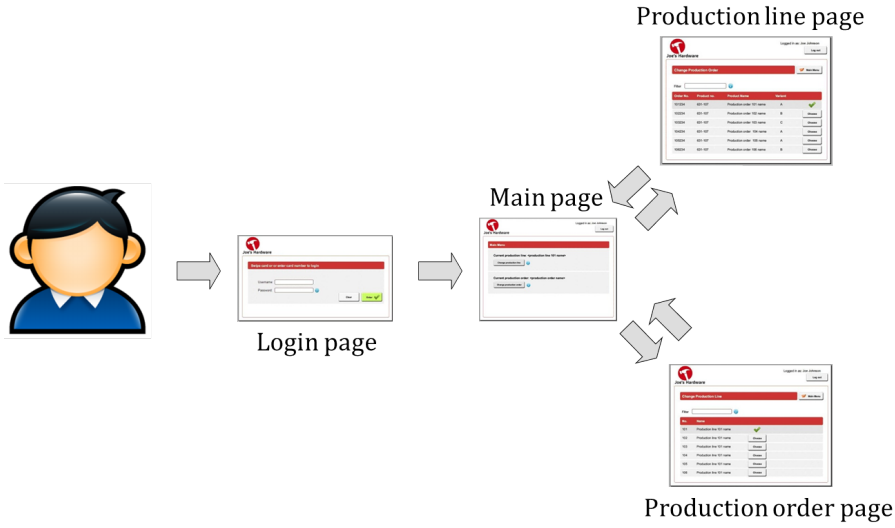


Figure 1.2: *Conveyor Operator* application.

error message. On the main page the conveyor operator is presented with two options: he can change the current production line or change the production order on the production line. If he chooses the first option, he is redirected to a page where he can switch between available production lines and change the sequence of the production line execution. If user chooses the second option, he is redirected to the production order page of the current production line where he can regulate the sequence of orders processed on this production line. On both of these pages the user is provided by the filtering options. Also both of these pages use drag and drop facilities to simplify the change of production line sequence and production order sequence on the production line, respectively.

1.4 Thesis structure

The rest of this thesis is structured as follows. In the Chapter 2, we discuss the state-of-the-art in Web application development as well as give an overview of the Model-Driven Development approach. In the Chapter 3, we describe design considerations and modeling notation of our modeling language. In the Chapter 4, we describe the technologies used for the implementation of the prototype tool and present the prototype tool we developed. Finally, conclusions and proposal for future work is given in Chapter 5. In the Appendix A the

the complete metamodel of the Domain Specific Language is presented and Appendix B contains a software user guide.

CHAPTER 2

State of the Art

In this chapter we present a discussion on the main characteristics of Rich Internet Applications and their improvements over traditional Web applications as well as give an overview of the main concepts of model driven engineering. The chapter is concluded by a analysis of engineering approaches for model-based development of Web applications. The theoretical background described in this chapter will be the driving force behind the development of our domain-specific modeling language, discussed in the Chapter 3.

2.1 Rich Internet Applications

The Web, originally, consisted of Web pages which had fixed content and, mainly, were educational institution and businesses profile pages. The problem with these Web sites was that the content did not change, unless manually updated by a webmaster. In course of time, Web pages became more dynamic, delivering Web pages created on the fly based on a content stored in databases. However, even though the content of these websites was generated in real-time they provided limited user interactivity and allowed only passive viewing of the content. It was because the page had to be regenerated on each user interaction, for example, change of the value in the dropdown list. As the Web technologies continued to evolve, there was found a way to enhance user experience to some

extent by utilizing scripting languages such as JavaScript and VBScript. Using these languages, the developer could write scripts for Web pages which could be invoked on the client (user's computer) once a page were loaded in a Web browser. These scripts improved interaction by immediately responding to the users' actions, executed fast because they didn't require a round-trip to a server, gave developers more control over the look and behavior of interface elements and improved overall usability of Web sites. However, the main disadvantage still remained - the underlying request/respond or so-called synchronous communication model used between client and server [22]. This model implies that all data resides on the server and a new Web page is generated for each user interaction, for instance, click on a button. There are several disadvantages of this model. First, any new data generation requires complete round-trip to the server which is inefficient. Secondly, the UI must be refreshed, even if it does not change. Lastly, user cannot continue to work with the application until the page is returned from the server. In Web Engineering this generation of websites is generally referred to as Web 1.0 [6].

Over the time, a new class of Web applications emerged as a result of development of several Web-related technologies and standards such as improved browsers, AJAX, REST, SOAP, Web Services, open APIs and more. This generation Web applications are grouped under the term Web 2.0. The use of new technologies provided users with an opportunity to become a dynamic content creator of a Web page in a form of text, video, and photo postings along with comments, tags, and ratings [21]. Basically, these applications provided users with ability to collaborate and share information online. At the same time, complex Web-based business systems started to appear, which involved a complex multi-step business workflows. This lead to development of a new type of Web application called Rich Internet Application (RIA). RIAs offer a series of advantages over traditional Web applications such as data distribution between client and server, distribution of page computation, improved client-server communication and more efficient user interfaces.

There is great diversity of definitions of RIAs. One of the most concise is a definition written by San Murugesan ([6], p.13):

"Rich Internet Applications are Web-based applications which run in the Web browser but have features and functionality which is comparable to traditional desktop applications."

This kind of application was developed to overcome the main limitation of the traditional Web application - its communication model. In contrast to traditional Web applications, the main advantage of RIAs is that some processing is transferred to the user's computer. For instance, if user clicks on a button only

the specific part of the page is refreshed without a need to contact the server and reload the whole page. This makes application to response faster and reduces network traffic. Other advantages of RIAs over traditional Web applications are discussed in details in the following sections.

2.1.1 Data distribution

In Web 1.0, an applications' data resides on a server, while in RIAs data can be distributed between both, server and client [3]. Typically, RIAs would transfer data needed for a user interface to a client while the application data would be kept on a server. This allows validating and preparing the data on client side without a need to contact the server [3]. Depending on data distribution pattern there can be distinguished two types of RIAs. The first type of RIAs is capable of running in a browser without any additional software installation and fetches additional data from a server as it is needed. For example, the server can be called for additional data as user interacts with interface widgets. Usually, this kind of applications are developed using AJAX technologies. The second type of RIAs is more advanced applications that can run out-of-browser in so-called sandbox or secure environment and can manipulate data without a need of contacting a server. Eventually, application has to communicate with a server, but it can be done when all operations on the client side are completed and data is ready to be sent to the server [3]. This kind of applications enables to load large sets of data into the background and therefore improve overall performance [24]. In contrast to RIAs developed with, for example, AJAX that does not require any plug-in installation, this type of RIAs require special runtime environment to be installed beforehand.

2.1.2 Distribution of page computation

One of the limitations of Web 1.0 applications is that page is the smallest unit of transfer or refresh [22]. This makes a multi-step business process to be divided into sequential steps where each of them is represented on its own page. This page-centric user interface results in slow and often frustrating user experience because he is forced to flip pages back and forth to be able to make any modifications or view data he had input [22]. To solve this problem RIAs introduce intermediate layer of logic between a client and a server, so-called client-side engine. This client-side engine is downloaded at the start of the session and is responsible for separation of user's interaction with the application from the browser's communication with the server [23]. In other words, the client-side engine acts as a mediator between client and server. One of the main purposes

of a client-side engine is to compute and refresh parts of the page [3]. For example, on a hotel reservation page the client-side engine would be responsible for updating hotel availability and pricing information depending on the country name the user selected from the country dropdown list. Employing client-side engine all these operations can be performed on a single page without a need to flip pages back and forth as it was needed in traditional Web applications.

2.1.3 Client-server communication

As mentioned earlier, traditional Web applications are based on a request/respond synchronous communication model. Each time user performs an action on the Web page, such as pressing submit button to submit a form or pressing on a link, the browser sends request to the server. When the server has processed the request, it responds and a new page is loaded by a client browser. The synchronous communication model is shown in Figure 2.1.

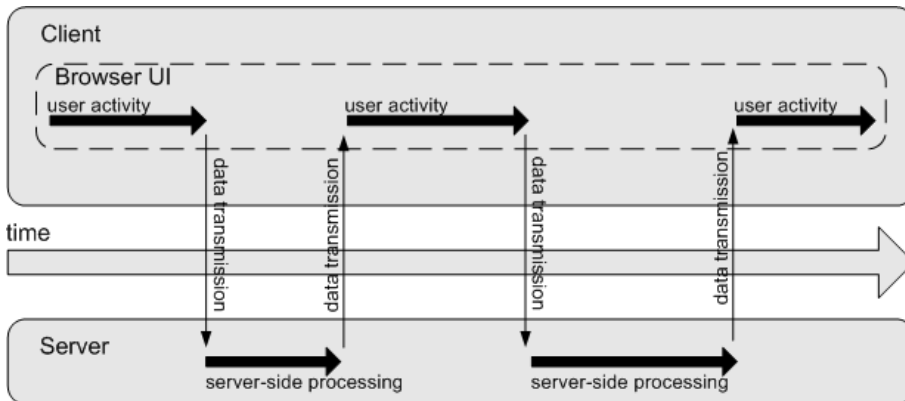


Figure 2.1: Synchronous communication model [26].

On the other hand, in RIAs by using client-side engine asynchronous communication can be achieved between client and server. This allows a client to interact with a server without having user to press a button or link. As soon as data is needed, it can be fetched from the server "behind the scenes". For example, if input data on the form needs to be validated, it can be done by the client-side engine. However, if client-side engine needs some extra information, it can asynchronously call the server without interrupting the process. The asynchronous communication model is shown in Figure 2.2. As it can be seen from the figure, the user will not feel any interruptions when using the application because the client-side engine will try to anticipate the users' actions and fetch the currently needed information from the server-side in advance.

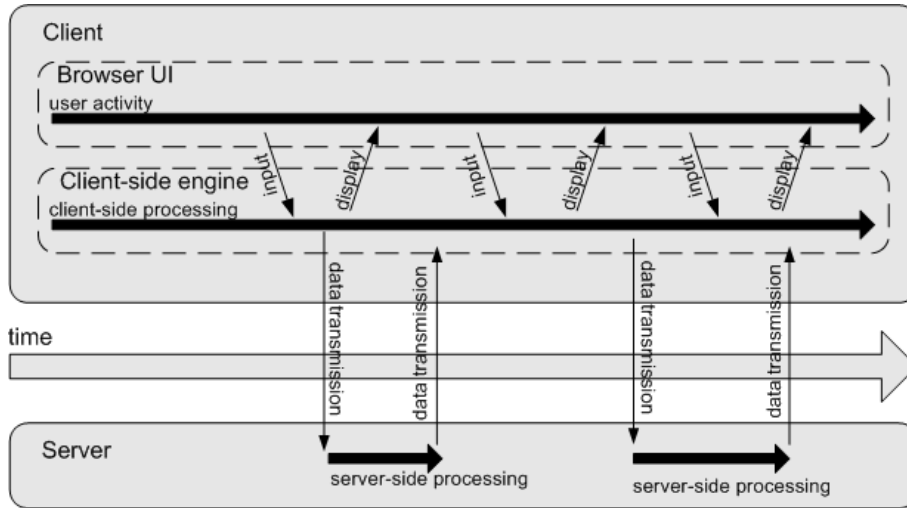


Figure 2.2: Asynchronous communication model [?].

This communication model greatly optimizes network usage and balance load on a server. In traditional Web 1.0 application servers had to process requests concurrently from many users. By introducing an asynchronous communication model the amount of computations on the server is decreased, thus allowing the same servers to handle more client sessions concurrently. The network usage is optimized as required data exchange between the client and server is reduced to only essential requests [23].

2.1.4 Rich user interface

Traditional Web applications have major limitations regarding the usability and interactivity of their user interfaces [3]. The word "rich" in RIA refers to improved responsiveness, usability and graphical richness of RIA user interface widgets in comparison to Web 1.0 generation applications. Using Web technologies such as Flash designers are able to create graphical user interfaces which resemble desktop applications. These interfaces, for example, can include such features as multi-windows, multi-tabs, menus, drag-and-drop and many more. RIAs are capable of visualizing complex data and can include different media, for example, audio, graphical elements, video etc. An example of RIA with a complex user interface is shown in Figure 2.3.

This is a website of Mini car manufacturer, available at <http://www.mini.com>.

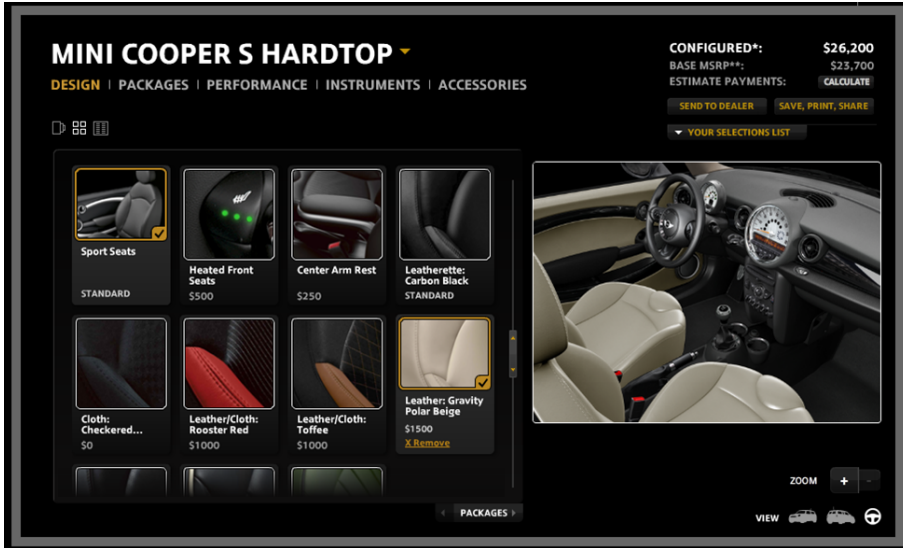


Figure 2.3: Mini official website screen dump.

On this website the user can make up a Mini car using different parts and preview the simulated look of the car right away. As it can be seen from the figure the main advantage of RIAs is that it eliminates multi-screen interface so common to Web 1.0 applications. All the operations are done on the same, single-screen view. This can be achieved by using the client-side engine which handles the events of the interface primitives such as mouse over, drag-and-drop and many more.

In this section we have discussed the state-of-the-art in the development of Web applications. The next section will discuss the MDD approach and its implementation frameworks.

2.2 Model-Driven Architecture

From the very beginning of the computer age, researchers in software engineering and developers have been working on defining higher levels of abstraction at which software is designed. Historically, raising the level of abstraction reduced the complexity of software development and improved productivity. Early computer programs were developed using raw machine code. The programming was long and tedious process since machine code was difficult to understand,

read, and remember. Over time, the first level of abstraction was achieved with the invention of assembly languages. These languages introduced a symbolic representation for machine instructions. The code became easier to read, but the problem with the assembly languages was that they were designed for specific processors. This drawback was eliminated by the introduction of procedural programming languages. These languages allowed moving away from the machine-centric way of computing towards a more application-centric way since a program could be written almost independently of the processor it would be running on [27]. Once again, the level of abstraction was raised. As software systems become larger and more complex, new software development approaches were required. This led to a creation of the third level of abstraction that was introduced with the emergence of object-oriented languages such as Smalltalk, C++, Object Pascal etc. Programs developed using these languages ran on a virtual machine that was responsible for converting the language byte code into machine language and executing it. It made possible programs written on a specific hardware platform to be ported to any other hardware platform on which the virtual machine was installed [27].

Today, software researchers are working on the transition to the next level of abstraction which would "address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively" [28]. This new approach is called *model-driven engineering*¹ (MDE). MDE is raising the level of abstraction by using models to describe the solution independently of the underlying programming language or specifics of execution platforms [27, 29]. A well-known model-driven engineering implementation framework is *model-driven architecture* (MDA) defined by Object Management Group (OMG). As stated in the OMG MDA guide ([32], p.58):

"The MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform."

Using this framework a developer starts with creation of software models independent of implementation technologies which are, gradually, transformed to specific technology platform models and, finally, the code. In more detail MDA and other related concepts are described in the next sections.

¹In the literature model-driven engineering is often called model-driven development (MDD) or model-driven software development (MDSD). Both of these terms refer to the general idea of MDE. However, MDE is wider in the scope than MDD and MDSD.

2.2.1 MDA Software Development Process

Throughout the history, humans have been creating models. Ancient Egyptian and Greek engineers, for example, were building ship models to study the designs and communicate them to others. Nowadays, models are used in different industries. For instance, mechanical engineers create models of automobiles, architects draw blueprints of buildings, electrical engineers build models of electric circuits etc. There are several reasons why engineers build models. Advantages of models are that they improve communication amongst different stakeholders, allow visualizing finished product before it is fully constructed and represent precise specifications of work to be done [35]. However, the main reason is that they are cheaper to build and change rather than the real thing.

It is not a secret that development of software is an expensive and complex process, which involves many different parties and consists of several stages. A traditional software development life cycle is shown in Figure 2.4. It includes a

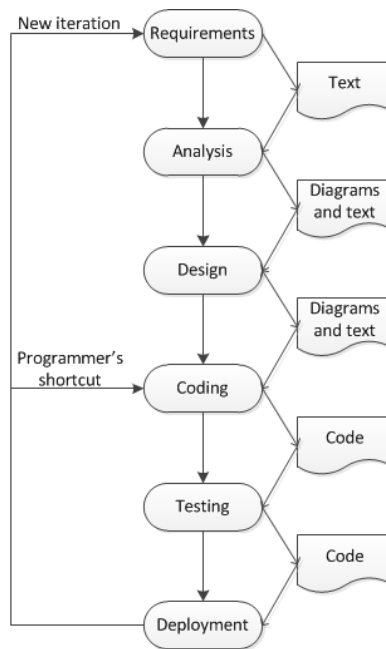


Figure 2.4: Traditional software development life cycle [31].

number of phases such as requirement gathering, analysis, design, coding, testing and deployment [31]. In traditional software development, software is usually modeled using different UML diagrams, which are produced during the first

three phases. These diagrams are used to express the details of the system being developed and serve as a reference when writing the code. However, the main issue is that when the design phase is done and the coding begins, the connection between the code and these models is gradually lost [31]. Changes are often done at the code level only and are not resembled in the design specification.

In turn, the model-driven software development approaches use models as the primary artifacts of the development. The development process with the MDA is shown in Figure 2.5. With the MDA the development process starts with

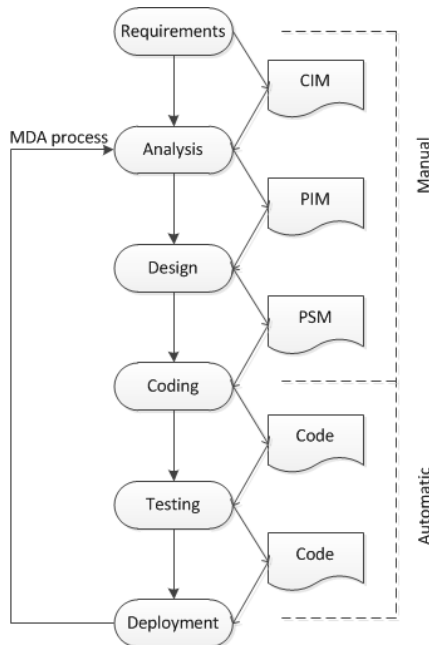


Figure 2.5: MDA software development life cycle [31].

a creation of computation independent model (CIM). The main focus of this model is on the environment of the system and on the specific requirements of the system. The CIM is further refined to platform independent model (PIM), which gathers all the information needed to describe the behavior of the system in a platform independent way. The platform specific model (PSM), in turn, represents the PIM taking into account the specific platform characteristics [36, 40]. In MDA the platform is defined as a "specific software implementation technology and/or specific hardware that constitutes the execution environment of the system" [31]. For instance, possible software platforms could be J2EE or .NET platform. As the last step, the application code is generated from the PSM. The traditional and MDA software development processes may look

similar. However, the main difference is that in the latter model to code as well as model-to-model transformations are performed automatically instead of performing them by hand. MDA provides a developer with an infrastructure for defining and executing transformations between models of various kinds [40]. There already exist tools for transforming PSM models to code and to some extent PIM models to PSM models. It is more difficult to transform CIM model to PIM model and, usually, PIM model is manually derived from the CIM model.

Before we continue discussion on the differences of the CIM, PIM and PSM models and their transformations we, first, have to define what the model is in terms of MDA.

2.2.2 Models

As mentioned earlier, modeling has a long history. The word model comes from a Latin word *modulus* (diminutive form of *modus*) which means "measure, standard". In the literature various definitions of the word model exist. Bezivin and Gerbe ([33], p. 273) give a following definition of the model:

"A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system."

Another definition which comes from Kleppe et.al ([31], p.15) describes a model in a more specific way directed towards the MDA:

"A model is a description of (part of) a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer."

Based on the definitions we define a model as an abstract representation of the real thing modeled using well-defined modeling language. In MDA, the Unified Modeling Language (UML) is used as a standard modeling language. The UML emerged from work in the 1990s on object-oriented notations which were integrated together to form one language [29]. Today, the UML is the standard modeling language of the software engineering. The UML 2.2 version consists of a set of 14 different diagram types that may be used to specify and design software systems. With the UML engineer can describe the static structure of the

system using elements such as classes, interfaces, objects, components, nodes and relationships as well as dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. Example of a structure diagram is the UML class diagram, which is used to model the structural aspects of the system, describing what attributes and behavior each object has. The example of the UML class diagram showing user hierarchy for a simple Web site is shown in Figure 2.6.

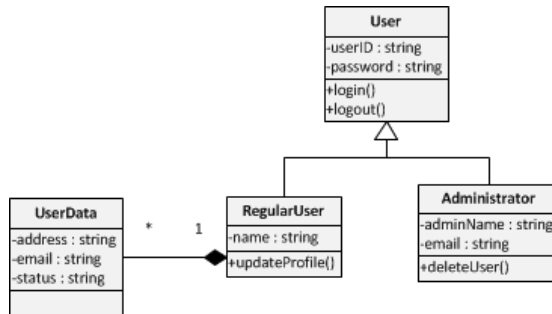


Figure 2.6: UML class diagram of simple Web site.

The diagram shows that there are two types of users available in the application. Each of them has its specific attributes and operations. Some attributes such as user id and password and operations such as login and logout are common for both user types.

To describe the behavior of the objects, dynamic diagrams such as UML activity diagram are used. An activity diagram describes the flow of control of the target system. Figure 2.7 describes a user login activity on the Web site. The diagram includes call actions that invoke other activities. For example, the Login action triggers ValidateCredentials activity and so on. The process shown in the diagram is fairly simple. When a user will tries to login his credentials will be validated. If the user exists in the system, he will be redirected to his account page; otherwise an error message will be displayed.

The use of common UML standard improves cooperation between technical and non-technical parties involved in the project and reduces the possibility of mis-interpretation of the models [31]. UML achieves it by providing the industry with standard mechanisms for visualizing, specifying, constructing, and documenting software systems. However, developers which use the MDA framework are not restricted to use only UML. Since UML is a general purpose modeling language it may be too large to be used for all domains. Therefore, developer may instead define his own domain specific modeling language. How this can

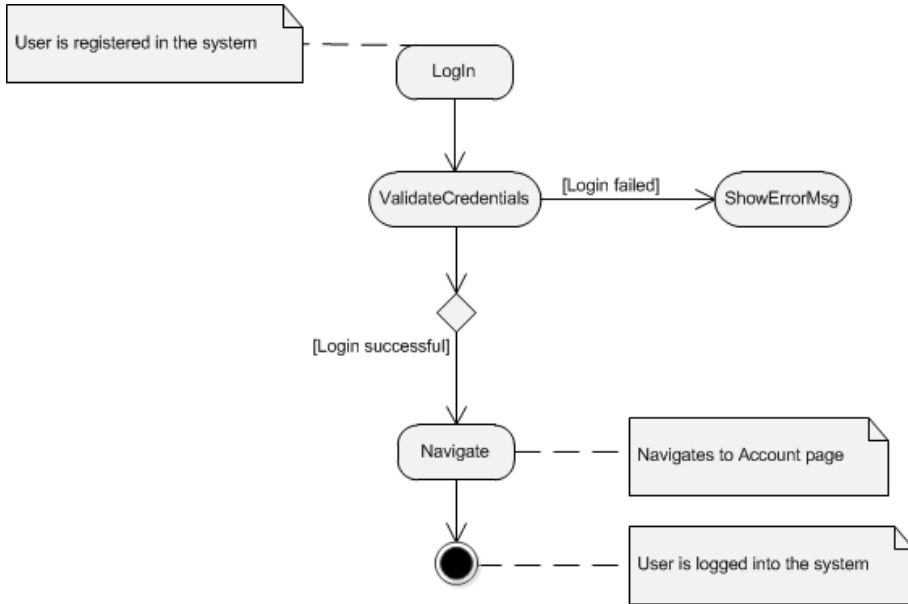


Figure 2.7: UML activity diagram of Login process.

be achieved in MDA framework is described in details in the next section.

2.2.3 Metamodels

The Greek prefix "meta-" means after or beyond and, therefore, a metamodel is often defined as a model of a model. A more accurate definition of a metamodel is following [34]:

"A meta-model is a model that defines the language for expressing a model."

A metamodel is used to define the structure and semantics of a class of models [31]. In other words, it defines what can be expressed in the valid models of a certain modeling language [42]. For example, the UML class diagram shown in Figure 2.6 is captured by a UML metamodel. It contains UML elements such as Class, Attribute, Operation etc. In Figure 2.8 a small subset of UML metamodel is shown.

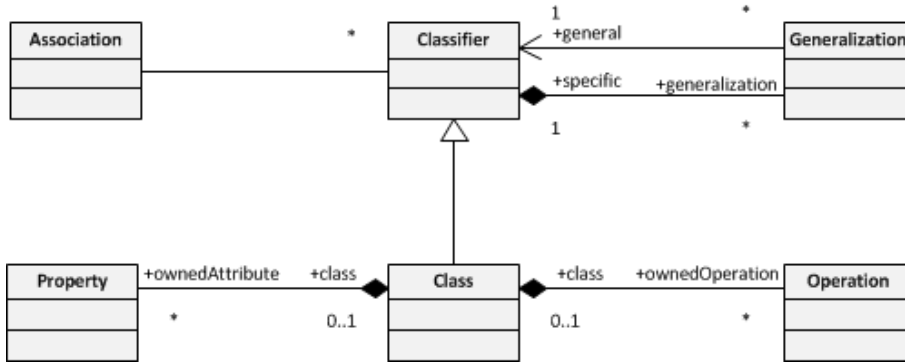


Figure 2.8: Excerpt from the UML metamodel [40].

Using the UML metamodel an architect can describe an object-oriented software system. However, there might be a need to define other metamodels, capturing, for example, domains like process, data warehouses, tests etc. [38] To be able to capture these different metamodels a meta-metamodel is introduced. It defines the core modeling concepts and is defined recursively, i.e., a model on this layer is an instance of itself. The OMG standard for defining metamodels and also a part of MDA framework is OMG Meta-Object Facility (MOF). The MOF emerged from the recognition that UML was one possible metamodel used in the software development, but that it was not the only one [38]. As the framework used for defining specific modeling languages, the MOF specification adopts four-layer architecture [40]. This architecture is explained in the next section.

2.2.4 MDA four-layered architecture

The traditional framework for meta-modeling is based on architecture with four layers. In OMG terminology these layers are called M0, M1, M2, M3. In theory there can be infinitive number of levels, but usually there is nothing beyond level M3, because it is self - describing [31]. In Figure 2.9 MDA four-layer architecture is depicted.

The M3 layer is the meta-metamodel layer. It describes concepts that appear in the metamodel. For example, the MOF. The M2 is the metamodel layer. It describes concepts that make up a modeling language. On this layer the UML metamodel or any other domain-specific metamodel is described. The M1 layer is the user model layer. It describes any model with a corresponding metamodel from M2 layer. For example, UML class diagram, activity diagrams etc. The last is the M0 layer or the data layer. On this layer there is a running system

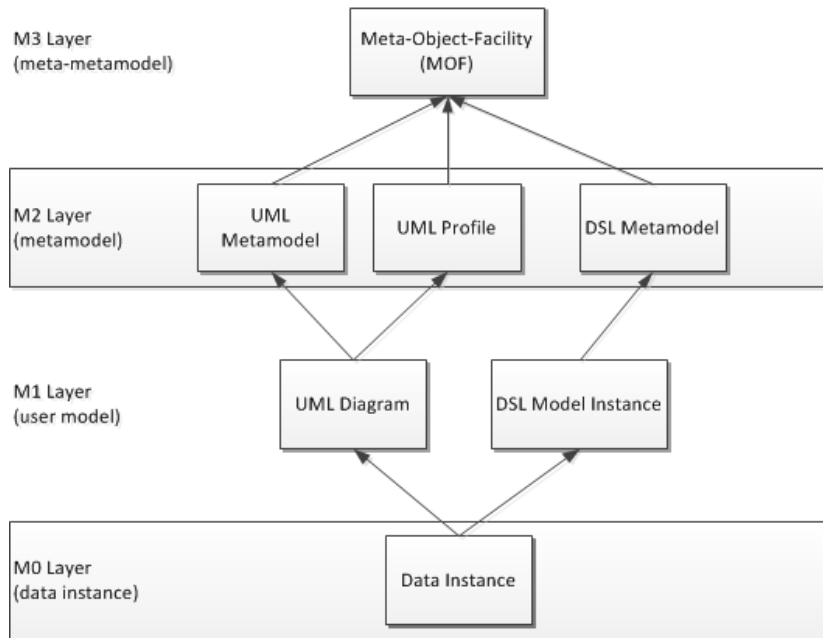


Figure 2.9: MDA four-layer architecture.

in which the actual instances exist.

To explain the four-layer architecture we will use a `RegularUser` class from the example shown in Figure 2.6, a simple user hierarchy model of a Web site. Figure 2.10 shows the example described with MOF. The lowest M0 layer contains the data of the application. In our example, these are `RegularUser` instances named "Jesper" and "Jens". On the M1 layer we define the concept of `RegularUser` with its properties and operations. The M2 layer defines the concepts needed to reason about concepts from layer M1, for example, a `Class` and an `Attribute`. Here we use the UML metamodel as a modeling language. On the M3 layer MOF metamodel is used to capture the structure and semantics of UML metamodel. An element at the M3 layer specifies the elements at the M2 layer. The same relationship is present between elements of layers M2 and M1 and M1 and M0, which is depicted by the instance-of relationship. As long as every element has a classifying metaelement through which metadata can be accessed, any model can be build and any system can be described [31]. All layer M3 elements conform to MOF since they are defined as instances of concepts of the M3 layer itself.

There are two main reasons of having the meta-metamodel in the MDA frame-

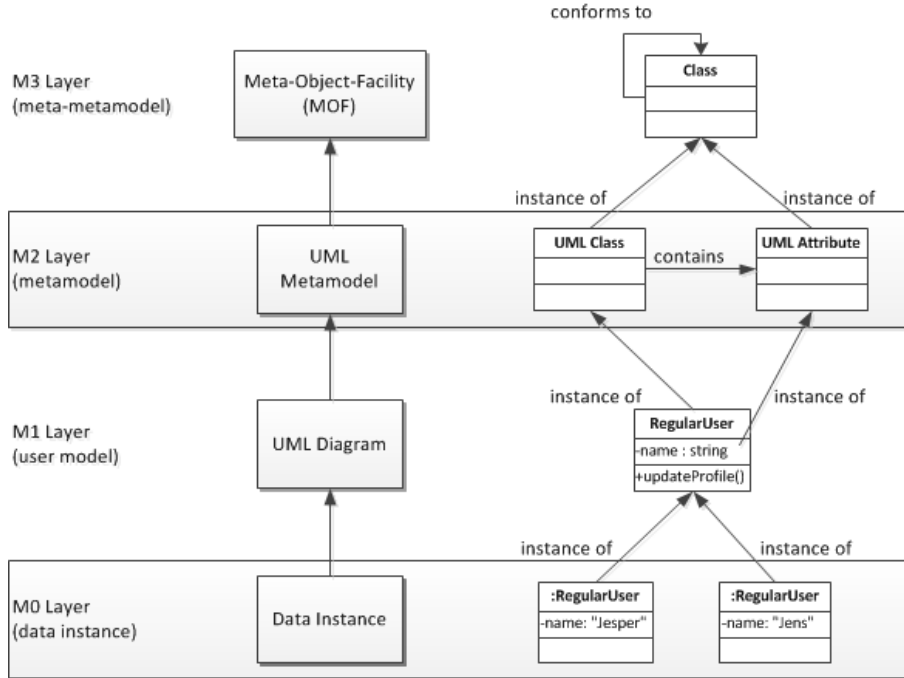


Figure 2.10: Overview of layers M0 to M3.

work. First, it is the mechanism for defining modeling languages. As Figure 2.9 suggests the MDA is not restricted to use the UML exclusively for all modeling activities. Since the UML might not be applicable for all problem domains, a developer could define his own Domain Specific Language (DSL). DSL would allow the developer to define solutions at the level of abstraction of the problem domain and use it for modeling tasks within the MDA framework. The concept of the DSL is described in the next section. Secondly, to be able to define the transformations between different models, the transformation rules have to be defined between a model in a source language and a model in a target language. These rules are defined based on a source and target language metamodels [31]. More details on the different model transformations are given in the section 2.2.7.

2.2.5 Domain Specific Language

In the previous section, we have discussed the MDA four-layer architecture which allows a developer to define a domain-specific modeling language by creating its

metamodel. In this section we will explain what a domain-specific language is and how it can be developed in the MDA framework.

The definition of the domain-specific language can be captured in the following statement [37]:

"A programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain."

General purpose languages (GPL) such as Java or C# are used to solve problems in a wide variety of application domains. In turn, domain-specific languages are usually built to solve problems in a specific domain. Well-known examples of DSLs are Structured Query Language (SQL), that is a standard language for accessing and manipulating databases, and Hypertext Markup Language (HTML) that is a markup language for Web pages. DSLs are usually defined to help creating the particular system more efficiently than existing languages would allow. Also, the use of DSL improves communication with domain experts.

In this thesis, we are going to create a domain-specific language, therefore we have to define what the main components of such a language are. A formal specification of a language normally consists of a structure definition (also known as abstract syntax), a representation definition (concrete syntax) and a definition of the semantics [30]. In order to explain the different domain language concepts we will use an example of simple Petri net shown in Figure 2.11 described as UML Object diagram.

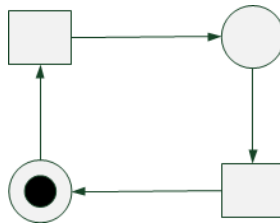


Figure 2.11: A Simple Petri net example [41].

The shown Petri net is composed of a number of distinct elements. Petri net consists of two types of nodes: places and transitions and an arc between them [41]. Arc can run only from a place to a transition or from a transition to a place. Places in a Petri net may contain zero or more marks called tokens.

Graphically, places, transitions, arcs, and tokens are represented respectively by: circles, squares, arrows, and black dots. All these different elements, as well as the way they are related, are defined in the scope of the Petri net metamodel which is shown in Figure 2.12.

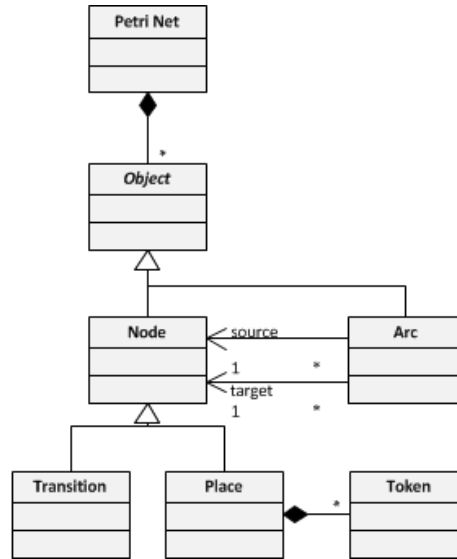


Figure 2.12: A metamodel for Petri nets [41].

The Petri net metamodel defines the abstract syntax of a language by describing the set of concepts provided by the language and how they may be combined to create models. Abstract syntax defines the structure of a language separated from its concrete notational symbols. In MDA the MOF is used to define metamodels and, thus, defines an abstract syntax of the modeling language. In Figure 2.13 the example shown in Figure 2.11 is described using an abstract syntax. It is described as the UML Object diagram.

The concrete syntax defines the graphical appearance of the language concepts. In order to define a graphical notation of the language a mapping between abstract syntax and concrete syntax is defined as shown in Figure 2.14.

The abstract syntax of the modeling language defines what models can be built. However, the meaning of the language is described by its semantics. In UML and MOF models, the Object Constraint Language (OCL) could be used for this purpose. OCL offers a formal notation to constrain the interpretation of model elements and consists of such constructs as precondition, postcondition and invariant [43, 40]. Using these constructs developer can define well-formedness

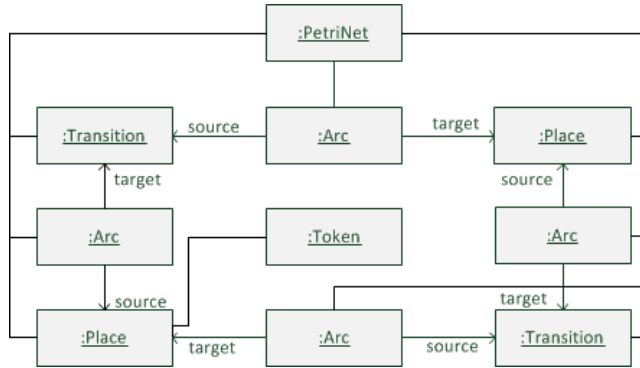


Figure 2.13: Simple Petri net abstract syntax [41].

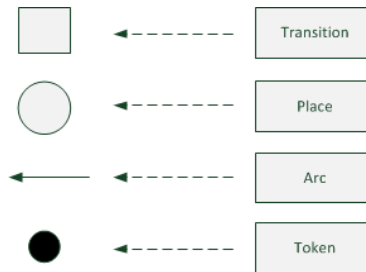


Figure 2.14: Mapping between abstract and concrete syntaxes [41].

rules that are the rules that specify a condition that a model must satisfy in order to be well-formed. OCL expressions are evaluated over a model, but they cannot change the model. The constraints are used to add precision to the model and address ambiguities in its specification. As described earlier in Petri net description, the arc can run only from a place to a transition or from a transition to a place. This restriction can be captured by specifying the OCL invariant for an Arc. An invariant is a condition that must hold true for all given instances of a particular model element [40]. The example of the OCL constraint is given below in Figure 2.15:

There are two approaches how to create a domain-specific language using the MDA framework. The first is to extend the UML metamodel using one or more profiles. The profile is an extension mechanism for adapting existing metamodel with constructs that are specific to a particular domain. Profiles are defined using stereotypes, tagged values and constraints that are applied to specific model elements, such as Classes, Attributes, Operations etc. For example, we could stereotype a "Class" RegularUser (shown in Figure 2.6) to be a "Person". The second approach is to create your own MOF-conforming domain-specific

```

context Arc inv:
  (self.source.ocliKindOf(Place)
and
  self.target.ocliKindOf(Transition))
or
  (self.source.ocliKindOf(Transition)
and
  self.target.ocliKindOf(Place))

```

Figure 2.15: The OCL rule for an Arc.

metamodel and use it to perform domain-specific tasks within MDA framework [35, 40]. There is no definite answer to which approach is better, that depends on the problem domain and experience of the developer.

In the next section an MDA framework as well as the role of DSL in this framework are described in more details.

2.2.6 Model Driven Architecture in Details

Model Driven Architecture (MDA) is a model-driven implementation framework developed by OMG and supported by the OMG core standards such as Unified Modeling Language (UML), Meta-Object Facility (MOF) and XML Metadata Interchange (XMI). The first two standards were already discussed in the previous sections. The XMI is a standard for exchanging metadata information via the Extensible Markup Language (XML). It can be used to exchange any model which metamodel is expressed in MOF [32, 45]. The use of these standards allows specifying modeling languages and models, and simplifies the exchange of the models between different UML CASE tools.

The main purpose of MDA is to provide programming language and execution platform independence. To achieve this MDA defines three levels of abstraction called viewpoints, from which a certain system can be analyzed. These viewpoints are computation independent model (CIM), platform independent model (PIM) and platform specific model (PSM) as shown in Figure 2.16. [45].

The CIM describes a business organization and system requirements using terminology that is familiar to the domain experts. CIM model focuses on the aspects of the business model that can be automated [27]. CIM model is often called a domain model. In order to explain the differences between different viewpoints we will use a simple banking Web site example. Consider the pro-

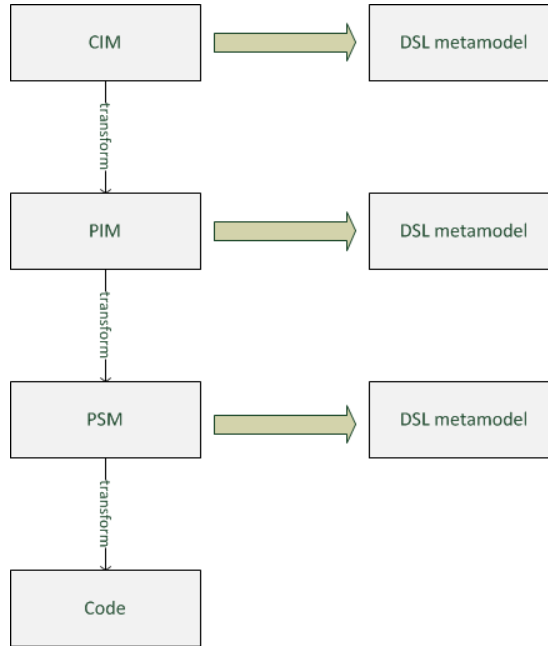


Figure 2.16: MDA classic approach [52].

cess of customer withdrawing money from his savings account. In this example we assume that the customer has sufficient funds in his account and exists in the system. The simplified CIM model of this process would look as follows:

1. Customer can log on the banking Web page using his username and password.
2. Customer can make online payment.

In this example, the CIM is used to model the requirements of the system.

The PIM is a model that is independent of any implementation technology. On this level it is not important whether system will be implemented on the J2EE or .Net or any other platform. Continuing our example, on this level we would define which operations have to be handled by a software system:

1. Web page verifies the customer identity.
2. The customer account is debited.

In this example, on the PIM level the data and process models would be defined.

The PSM is a model that relies on details that are important to the implementation of the system on a given platform [36, 40]. Usually, for each specific technology a separate PSM is generated [31]. The PSM would contain the same parts of the process as PIM, but implemented in a specific technology. In this example it would be the Oracle database and Web Service Definition Language (WSDL) models:

1. An Oracle database is contacted to verify customer's account data.
2. A Web service is called to subtract the asked sum from the customer's balance.

The final step in the development process is the PSM transformation to code. Since the PSM is already very close to the implementation technology the transformation to code is relatively straightforward. The MDA defines the CIM, PIM, PSM and code levels and also defines how they relate to each other. As it can be seen from Figure 2.16, each higher level starting from the CIM is gradually refined, with each level closer to the concrete implementation technology. It also shows that on each of the abstraction levels developer can define his own domain-specific language. It can be the same DSL for all abstraction levels or different DSL for each level with relationships defined between them.

The main idea of MDA is that transformations between the models are made automatically. Of course, not the whole process can be performed automatically, but there exist tools to transform the PSM to code automatically and there exist semi-automatic approaches to transform PIMs to PSMs. By semi-automatic we mean that there is a need to add some additional information manually. Transformations from CIMs to PIMs are usually done manually.

As stated by Kleppe et al. [31] the creation of transformations between different models is the hardest activity when developing software within the MDA framework. To eliminate the problem of having to define transformations between each MDA abstraction level an alternative MDA development approach is suggested, so-called incremental MDA approach. This approach is depicted in Figure 2.17

The difference between the classical and incremental approaches is that in the latter CIM, PIM and PSM extend each other in order to add more specific information on each abstraction level. The incremental MDA solves the model inconsistency problem which might appear in the classical MDA approach as a result of the model transformations between CIM, PIM and PSM levels. In

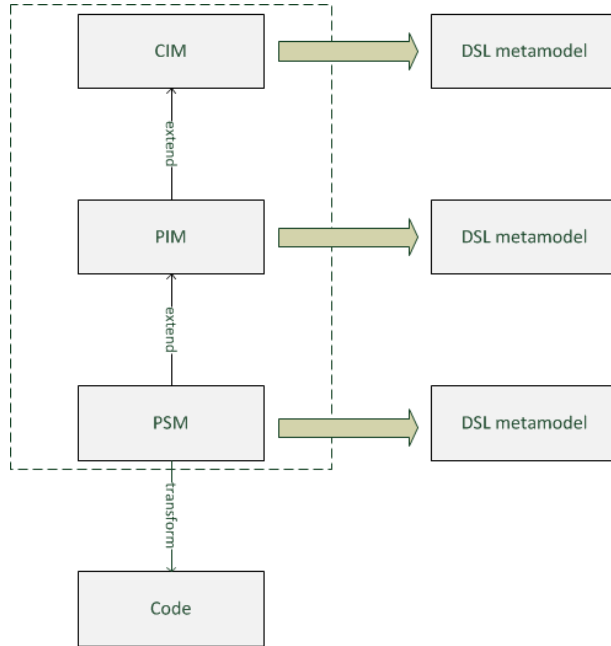


Figure 2.17: MDA incremental approach [?].

contrast to classical MDA, this approach does not require defining the transformations between all the layers, but only the transformation from the extended model to code.

In the next section we will look into different model transformations used in the MDA.

2.2.7 Transformation and Code generation

One of the most important concepts of MDA is the concept of transformations. The definition of transformation in MDA is captured in the following statement ([31], p.41):

”Transformation is the automatic generation of a target model from source model according to transformation definition.”

In order to transform models a transformation definition has to be defined. The transformation definition is as follows ([31], p.41):

"A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language."

The transformation tool is using transformation definition in order to perform transformations. In MDA model transformations can be of two kinds, mainly, model-to-model (M2M) and model-to-text (M2T) transformations.

Model-to-model (M2M) transformations are used to transform existing model to into a new target model. To be able to transform model-to-model one has to follow a set of transformation rules which define how one or more constructs in the source language can be transformed into one or more constructs in the target language [31]. Basically, developer has to define a mapping between the meta-model elements of the source and target languages. Since both source and target meta-models are expressed in MOF, it is possible to define transformation definition. Using this transformation definition a transformation tool will be able to transform one model instance to another. The general picture of this approach is given in Figure 2.18. Here the DSL metamodel is a source metamodel and C# metamodel is a target metamodel. The transformations between models can be achieved using XSLT language, MOF QVT, ATL and other transformation engines.

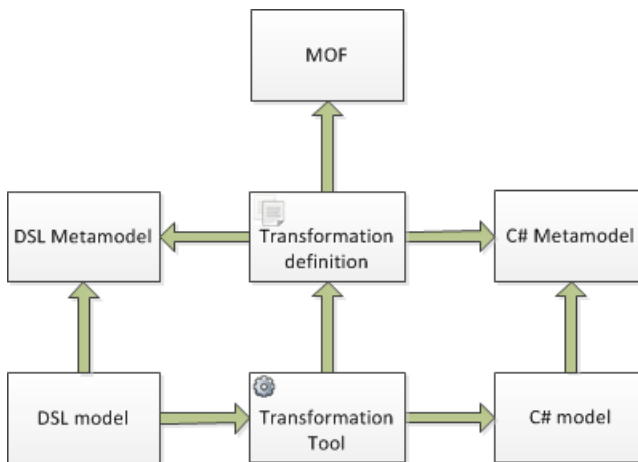


Figure 2.18: MDA model-to-model transformation.

Model-to-code transformations are used to transform models into a desired programming language. Code generation is focused on transforming existing model into an implementation in a specific target platform that could be executed. The transformations from a model-to-text can be done using scripting languages such as Java Server Pages (JSP), Jython or template based languages such as Xpand, MOFScript, Acceleo etc. Scripting languages like JSP allow generating code based on requests to the metamodel. Template languages such as XPand define template documents where certain parts are replaced with information from the model. Figure 2.19 depicts the process of model-to-text transformation.

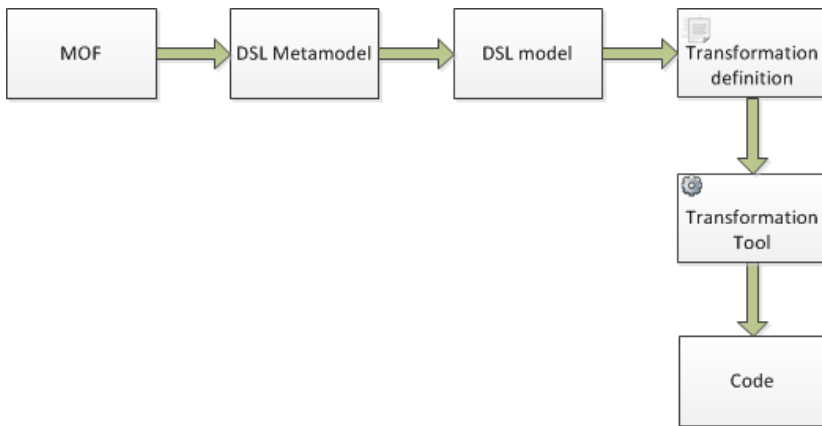


Figure 2.19: MDA model-to-text transformation.

Figure 2.20 shows the common transformation flow between the models in the classical MDA approach. As it can be seen from Figure 2.20 this approach requires several model-to-model transformations and, finally, model-to-text transformation which is used to generate the code from the PSM model. As mentioned before, there exists an alternative MDA approach, so called incremental MDA. In this approach only transformation from extended model to code is required. The transformation flow in this approach is depicted in Figure 2.21.

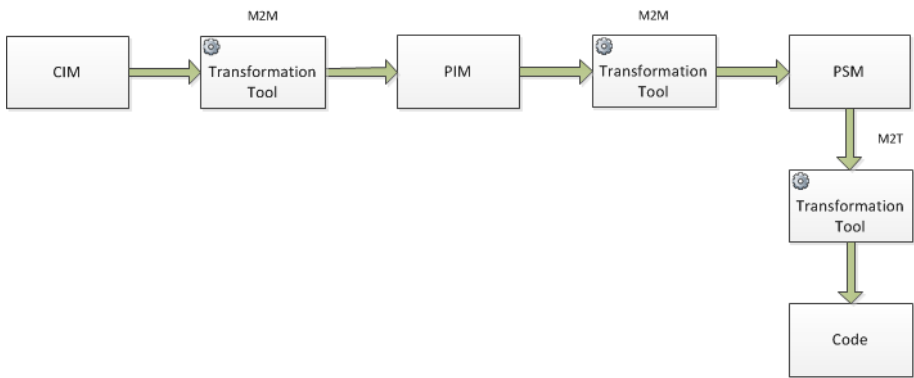


Figure 2.20: Model transformation flow in classical MDA.

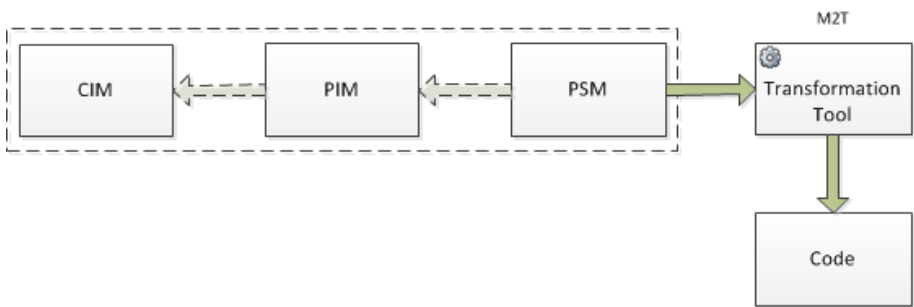


Figure 2.21: Model transformation flow in incremental MDA.

2.2.8 MDA chosen approach

In our solution, we decided to use the incremental MDA approach. As described earlier, it defines three levels of abstraction CIM, PIM and PSM which extend each other in order to add more specific information on each abstraction level. In our solution, we will create a Domain Specific Language for each of these levels. On the CIM level we are going to create a GUI model which will be used to capture the initial requirements of the Web application. On the PIM level it will be extended by additional layout, process and style models. Lastly, on the PSM level the layout model will be extended by the ASP.NET model. In order to transform models into the code we will define the model-to-text transformation which will be used to generate the code based on the PSM model. In more details the design of the modeling language and its prototype tool implementation is discussed in the following chapters.

In this section we have discussed the Model Driven Development approach. In the next section we present an overview of several Web modeling languages.

2.3 Study of Web engineering methods

In this section, we present an analysis of the methodologies for Web application design and development. Based on the overview of the Web application modeling methods by Shwinger et al. [63] we have selected several methods to analyze. These methods were selected because they include requirements modeling in the software development process. We chose 4 methods, namely, OOHDM, UWE, OOH and WebML. The conducted study covers the following points:

- Method description. It provides a general description of the method.
- Requirement specification techniques. It presents techniques method uses in the requirement specification process.
- Development process. It presents the models developed in the methods' design phase.
- Tool support. It discusses the availability of the tools supporting the method.

2.3.1 OOHDM

Method description. The Object-Oriented Hypermedia Design Model (OOHDM) is one of the first approaches which provided the methodology for the development of Web applications. It was created by Schwabe and Rossi in 1994. This approach is based on the object-oriented concepts. Using the OOHDM a developer can build Web applications using an incremental or prototype process model [57, 59]. The approach emphasizes the separation of different concerns of the Web application such as content, navigation and presentation and defines separate models focused on each of these design aspects [59].

Requirement specification techniques. The requirement gathering phase is divided into three steps:

1. Identification of actors
2. Use case specification
3. User interaction diagram construction

In the first step different user profiles as well as tasks users will accomplish while using the application are identified. Based on these profiles different navigational models can be built. In the next step usage scenarios are described. A scenario represents the set of subtasks the user has to perform to complete the tasks. Scenarios are specified textually. Usually, they are obtained from a sample of real users, who represent the intended audience or they can be defined by the developers. After collecting several scenarios a generalization is captured in a use case [57, 60]. In the last step of the requirement gathering process the use cases are described using User Interaction Diagram (UID). This diagram captures the flow of information and helps detail the information items and choices made by the user.

Design process. The process in OOHDM is divided in three phases producing the following results:

- Conceptual model
- Navigational model
- Abstract interface model

During a conceptual design phase a conceptual model of the application is build. It is represented as a class model and shows the static aspect of the system [13].

In the navigational design phase a navigational model is defined which is a view of the conceptual model. Different navigational models can be built for the same conceptual model based on different user profiles [59]. The navigational model is expressed in two diagrams: the navigational class diagram and the navigation context diagram [57]. The navigable objects of the Web application are defined by a navigational class diagram. The navigational context diagram is used to add a structure to the navigational class diagram. Context classes complement the definition of a navigational class by indicating which information is shown and which information is reachable when accessing the object in particular context [57]. The next step is a creation of an abstract interface model. In this model the associations between and interface objects and navigational objects are created. The last is the implementation phase. In this phase the navigational and abstract interface models are mapped into concrete objects available in the chosen implementation environment.

Tool support. The only available tool related to OOHDM is called HyperDE¹.

2.3.2 OO-H

Method description. The OOH stands for Object Oriented Hypertext method. This method was developed in the 2000 by Jaime Gomez, Christina Cachero and Oscar Pastor. It is partially based on the standards such as UML and OCL.

Requirement specification techniques. To capture the user requirement the OOH method uses the UML use case diagrams. The method uses it as a basis on which the navigational requirements are structured.

Design process. OOH is using three different models in its modeling process:

- Conceptual model
- Navigational model
- Presentation model

After the requirements have been specified the next step is creation of the conceptual model. In the conceptual design phase the conceptual model is defined by means of the UML class diagram which describes the structure of a system. In the navigational design the Navigational Access Diagram (NAD) is created. This diagram defines a navigational view which enriches the domain view with

¹<http://www.tecweb.inf.puc-rio.br/hyperde>

navigation and interaction features [61]. A developer should define a separate NAD for each separate view in the system as well as provide a different NAD for each user type who is allowed to navigate through the system. The last, presentation design phase consists of two models, an abstract presentation model and a layout composition model. Abstract Presentation Diagram (APD) gathers the concepts related to presentation. Composite Layout Diagram (CLD) gathers the concepts related to abstract structure of the site and specific presentation details. In this model the location and visual style of elements can be edited, widgets (implementation constructs) can be specified and new elements can be added to improve the visual impact of the generated interface.

Tool support. This method is supported by a modeling environment called VisualWade¹. This tool provides a set of model compilers that allow generating a running application for different platforms and languages [65].

2.3.3 WebML

Method description. Web Modelling Language or WebML has been developed by Piero Fraternali and Stephano Ceri from 2000. WebML language offer notation by using both Entity-Relationship and UML and graphical representation using UML syntax. This language is continuously improved and extended for a range of concepts such as web services, exceptions, process modeling and AJAX [62]. WebML provides a platform independent conceptual model for rich data web applications [62].

Requirement specification techniques. Requirement gathering process in WebML starts with user identification and personalization needs [13]. Requirement specification consists of classical UML use case diagram specification supplemented by a semi-structured textual description. To express more complex use cases the activity diagrams can be used.

Development process. The following five models are created during the design process in WebML:

- structural model
- composition model
- navigation model
- presentation model

¹http://gplsi.dlsi.ua.es/iwad/ooh_project/cawetool.htm

- personalization model

When the requirements are specified the development process in WebML starts with definition of the structural model which is an Entity-Relationship (ER) model of the data in the system and their relationships. The next step is a definition of hypertext model which is a combination of composition and navigation model constructs. Hypertext model is the most important in WebML and is used to represent a website [62]. The composition model describes the pages that compose the website. The navigational model provides contextual links between content units. The last model to be specified is the presentation model which defines the layout and graphic requirements for page rendering. The last personalization model defines the customization features.

Tool support. WebML is supported by the WebRatio¹ tool, a commercial CASE tool applied in an industrial environment [62].

2.3.4 UWE

Method description. UML-based Web Engineering or UWE is an object-oriented method (together with OOH and OOHDM). UWE was developed by Nora Koch in 1998. The approach consists of several models and uses UML for all models that are developed.

Requirement specification techniques. The first step toward developing a Web application with UWE is creation of requirements model. The UWE proposes to use the UML use case diagrams to capture the rough description of the functionalities. In addition to the UML features, UWE distinguishes among three types of use cases: navigation, process, and personalized use cases. The navigation use cases are used to model user behavior when interacting with a Web application, the process use cases are used to describe business tasks that end users will perform with the system and the personalized use cases are those that imply personalization of a Web system [64]. These diagrams can be complemented with documentation describing the users of the application, the adaptation rules and the interfaces [13]. The next step in requirement modeling is to create a more detailed description of the nontrivial use cases which is done using the UML activity diagrams [64].

Design process. The following four models are created during the design process in UWE:

¹<http://www.webratio.com>

- Content model
- Navigation model
- Process model
- Presentation model

Analysis models provide the basis for the design models, in particular the content model of a Web system. The content model is the UML class diagram describing the data structure of the application [64]. Based on the requirements analysis and the content modeling, the navigation structure of a Web application is modeled. The navigation structure model defines which classes of the content model can be visited by navigation [63]. In a next step, the navigation structure is extended by process classes that represent the entry and exit points to business processes [64]. Each process class included in the navigation model is refined into a process model consisting of a process flow model in the form of a UML activity diagram and optionally of a process structure model. In the next step the presentation model is built which abstractly specifies the structure of an element in the navigation model. The advantage of the presentation model is that it is independent of the actual implementation technologies. The last step is transformation of the models into platform-specific implementation.

Tool support. UWE is supported by the CASE tool ArgoUWE¹ that is based on the open source Argouml² software. In order to make development process more incremental, UWE supplies automatic/semi-automatic tool that allow generating most models from previous models automatically. Models generations are based on model transformation techniques (QVT and ATL).

2.3.5 Conclusions

We have presented a brief analysis of the several approaches for the development of Web applications. We focused on the requirement specification techniques that these approaches propose. Based on this analysis we can conclude that:

- The most popular technique for the specification of requirements is use case modeling. This is not surprising since are well-known approach for graphical representation and description of requirements.

¹<http://uwe.pst.ifi.lmu.de/toolargoUWE.html>

²<http://argouml.tigris.org/>

- Several approaches complement use cases with additional techniques such as the UML activity diagrams in order to capture nontrivial use cases.
- All approaches provide a narrow view on the navigational requirements since they focus on the use cases individually.
- All approaches are supported by a tool.

Modeling Notation

In the previous chapter we discussed the state-of-art in Web application development as well as described the main concepts of model driven engineering and its different implementation frameworks. Based on this theoretical knowledge we define the domain-specific language for model based development of interactive Web applications. This chapter is devoted to describe the design considerations and notation of the language.

3.1 Overview

Several model-driven approaches for development of Web applications exist. However, only few of them define a requirements model which is integrated with the lower-level models such as data or process models [52]. Most of these approaches use classical UML diagrams such as use cases and then use them to derive other models. In turn, we designed a domain-specific language which allows specifying requirements for the Web application using graphical user interface (GUI) models. These GUI models can be complemented by additional information, then transformed into concrete implementation platform model and the respective code can be automatically generated.

To develop the modeling language it was decided to use the MDA incremental

approach. By using this approach we are able to divide the domain-specific language into several abstraction levels. Each of these levels will extend another level with more specific information. This approach has several benefits. First, we do not have to define model-to-model transformations between each abstraction layer as opposed to the classical MDA approach. It reduces the development effort in the case of likely changes. Secondly, the associations between elements of different layers of abstraction are defined much clearer.

As mentioned earlier, different abstraction levels in the MDA framework allow separating different concerns of the software application. The MDA framework defines three levels of abstraction Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). This allows Web application to be modeled from different viewpoints and by different people. In our solution, on the CIM level, it will be possible to define the static view of the graphical user interface (GUI) which can be done by a business consultant together with a customer. On the PIM level process specification as well as layout and style information for the GUI elements can be added by a developer and web designer, respectively. On the PSM level, the PIM elements are mapped to the concrete ASP.NET platform elements. This task can be also done by a developer. The last task is the code generation which will be done automatically based on the PSM model. Figure 3.1 shows the architecture of the solution.

In more details the design solution for each layer of abstraction is discussed in the next section.

3.2 Desing of the Domain Specific Language

According to Koch et. al [13] and Aguilar et.al [46], there are not many Web engineering approaches which use prototyping as a tool for requirements elicitation or specification. In fact, Koch et.al [13] mentions only one approach called Design-driven Requirements Elicitation [16]. The approach defines methodology for developing Web applications which suggests using customer-developer joint exploration of design prototypes in order to specify the requirements of the application. Creators of this approach formalize the hypothesis that "the design process is critical in supporting not only the identification of client requirements, but also in the actual formulation of their needs [16]." This hypothesis is supported by empirical data obtained from the companies that specializes in Web development. However, the Design-driven Requirements Elicitation approach is defined only theoretically and is not supported by a modeling language.

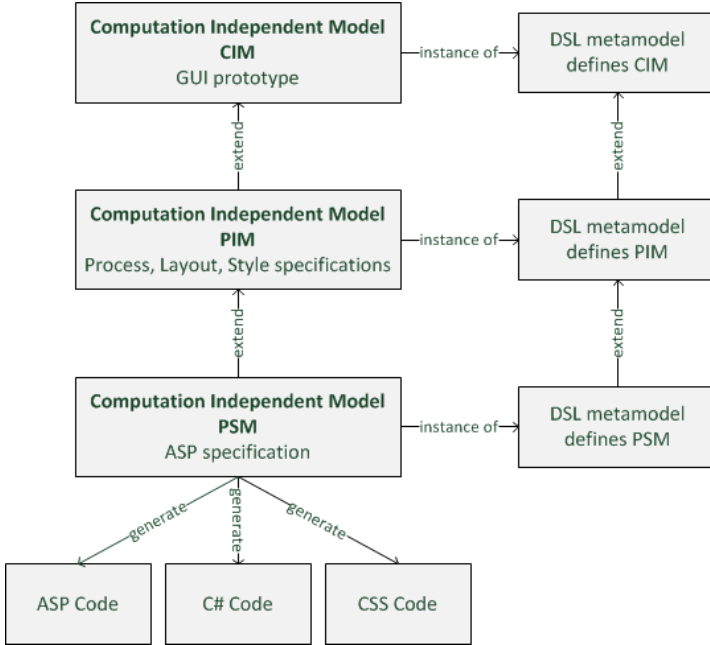


Figure 3.1: Architecture of the Domain Specific Language for interactive Web applications.

In our solution, we will model requirements of the Web application in the Computation Independent Model. Based on the aforementioned findings we decided to model graphical user interface (GUI) prototypes in order to capture the requirements of the application. However, as described in the Chapter 2, Web 2.0 applications provide more interactivity than Web 1.0 applications. In Web 2.0 application, for example, user interface elements can trigger events and use data. This means that the modeling language should be able to model processes that are invoked by the user interaction on the GUI. As discussed in the Chapter 2, one way of capturing the dynamic behavior of the software system could be to use the UML activity diagrams. In order to capture the dynamic behavior in our domain-specific language we have defined a process specification metamodel. This metamodel is a simplified version of the UML Activity diagram metamodel.

The Platform Independent Model will be used to create the process, layout and style specifications. The process specification defines the process which is triggered by an interaction with a user interface element. On the CIM level business consultant would only specify the name, trigger and description of the process. On the PIM level the concrete process model is defined by a developer.

Similarly, since the GUI defined by the business consultant defines only abstract view of the GUI it has to be refined further which is done by a designer using layout and style specifications. On the CIM level only the name and description is specified to describe the layout and the style of the GUI elements. On the PIM level, the layout specification is used to define a layout for the GUI elements and style specification defines the appearance of different GUI elements.

The Platform Specific Model includes information specific to .NET platform. Specifically, on this level the ASP model is defined which is used to specify the concrete implementation of the GUI elements. For example, in the GUI model the same GUI control can be used to represent such ASP controls as a dropdown list or a radio button list. Therefore, based on the ASP model we will know which control code to generate.

As mentioned earlier, the models can be extended by different people and even developed independently, helping to separate different concerns of the Web application such as requirements specification, dynamic behavior and presentation. To achieve this, the Domain Specific Language is divided in different abstraction layers. Each abstraction level is described more specific in the next section.

3.3 Domain Specific Language Metamodel

In the following sections, we will describe in details the Domain Specific Language we created for each of the MDA abstraction levels.

3.3.1 DSL Metamodel for CIM

On the CIM level we created a metamodel for the requirements of the Web application, which is the GUI metamodel.

GUI metamodel

In order to create a domain-specific metamodel for the graphical user interface we analyzed the Windows User Experience Interaction Guidelines [47], Apple User Experience Guidelines [48], GNOME Human Interface Guidelines 2.2.2 [49] as well as HTML 5 Specification [50] and the most common GUI elements were selected.

At this abstraction level we do not need a detailed model of the user interface presentation, but only to know what kind of components make up the user interface and how they may be grouped. A *GUIControl* defines what types of user interface controls are available in the model. As it can be seen from Figure 3.2, there are two subclasses of the GUI controls the *AtomicControl* and *CompoundControl*.

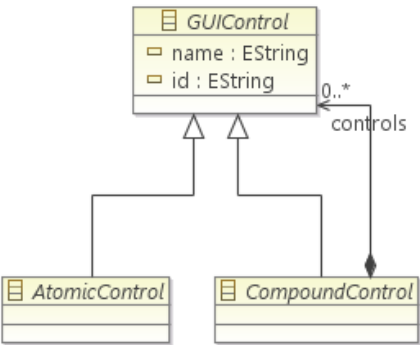


Figure 3.2: DSL metamodel for the *GUIControl*.

An *AtomicControl* is used to define a basic user interface controls such as Button, Field, Selector, Text and Image. All possible atomic controls are shown in Figure 3.3.

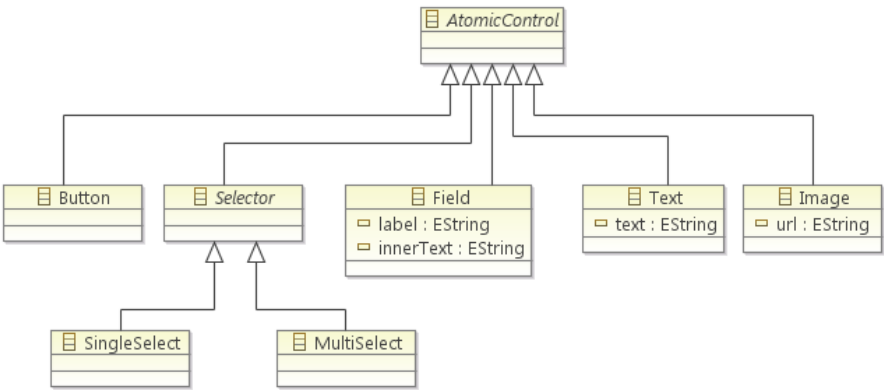


Figure 3.3: DSL metamodel for the *AtomicControl* controls.

A *Button* is a control which, when pressed, can trigger an action. A *Field* is

an area used to display text or where free-form text can be typed by the user. A Selector is a component which can be clicked upon to select or deselect an option. There are two types of selectors: the SingleSelect and the MultiSelect. MultiSelect is used to model an exclusive selector, i.e., the one that allows multiple options to be selected from the list of options. SingleSelect is a control which provides user with a list of choices from a specifically defined range and only one of option can be selected. A Text control represents static text that the user cannot edit. Image is an area containing an image.

The Web GUIs might also contain elements which are used to group other elements in a logical way. For example, in HTML the form or the table might act as a container for other elements. In GUI metamodel, we designed this type of elements using the Composite pattern [51] as shown in Figure 3.4.

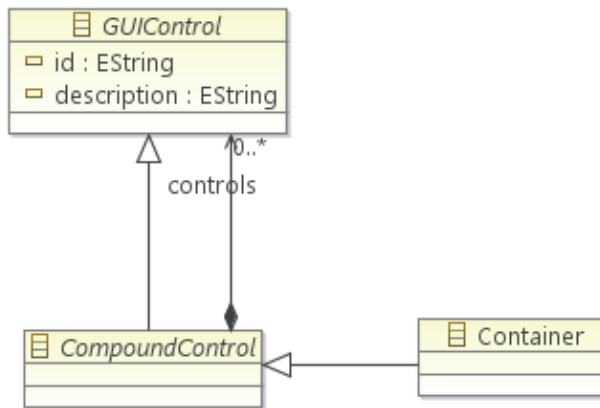


Figure 3.4: DSL metamodel for *CompoundControl* controls.

It can be seen that an abstract *CompoundControl* is used as a container for basic controls or it can contain other *CompoundControl* which allows modeling a nested hierarchy of containers. In our metamodel we have defined only one *CompoundControl* which is a *Container*. However, with current design it is very easy to introduce new *CompoundControls*, if needed, by simply adding a new class inheriting from the *CompoundControl* class.

As mentioned earlier, any GUI control can be associated with a dynamic behavior which, in the metamodel, is shown as a containment relationship to a process it triggers. On the CIM level, the business consultant has to specify only the name, the trigger and the textual description of the process. In the similar way for each GUI control a style and layout information can be defined.

On the CIM level only a name and a description of them have to be specified. This is depicted in Figure 3.5.

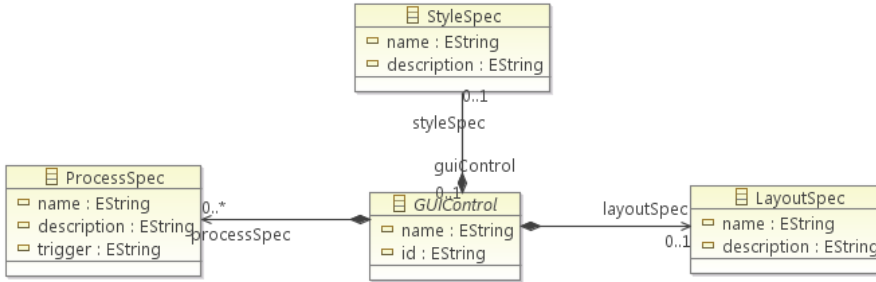


Figure 3.5: DSL metamodel of the *GUIControl* specifications.

The process, layout and style specifications are defined on a PIM level. The metamodel of each of them is described in the next section.

3.4 DSL Metamodel for PIM

On the PIM level we created metamodels for the process specification, as well as GUI control layout and style specification.

Process metamodel

The process model represents the process to be executed. The process can be seen as a container for all other activities. Since every process should have a start and end node we do not need to model them. To model activities in the process we used a block strategy by nesting action and control activities inside blocks. In this way all the activities in the process are properly nested.

The processes are triggered by an interaction with the GUI. The Process can consist of two types of activities: Action and Control as shown in Figure 3.6.

As the name suggests, control activities are used to control the execution of activities. From Figure 3.7 it can be seen that there are two types of control activities: Decision and Loop. To model Control activities we also used the Composite pattern. It allows expressing that a Loop and Decision activities can

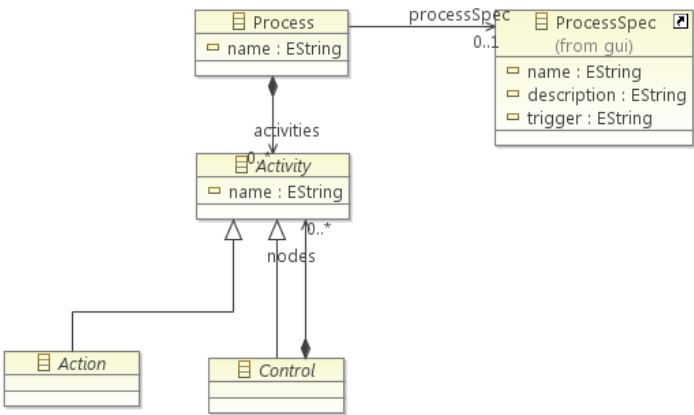


Figure 3.6: DSL metamodel for the *Process* specification.

contain another activities inside them such as Action or other Control activities. The Decision activity is used to model a conditional execution of the activities.

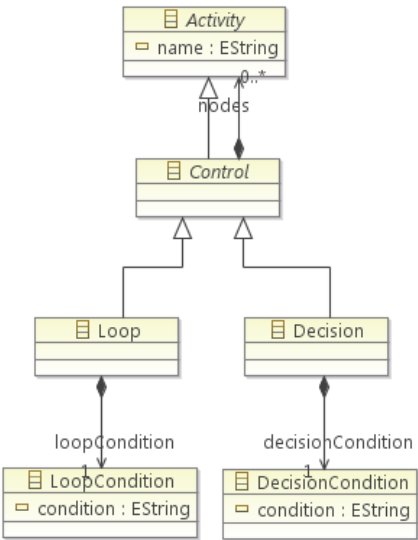


Figure 3.7: DSL metamodel for the *Control* activity.

The Loop is used to model the execution of the specified activities as long as the condition is true. Both Decision and Loop can be specified one Condition,

which is represented as a String. We used a String for the Condition to make the implementation of the modeling language simpler. However, the model could be easily extended with a condition expression model. The legal string for the condition, assuming that there exist a checkbox with id `chkCountry` could be `chkCountry.Checked` which would evaluate to true if the checkbox was checked. If the Condition of a Loop evaluates to true all activities in the Loop will be executed repeatedly while condition holds. In case of a Decision, if condition evaluates to true the activities on the true branch are executed, otherwise the activities on the false branch are executed.

The other type of activity is Action. This activity represents the units of work composing a process. All possible actions are shown in Figure 3.8. The Nav-

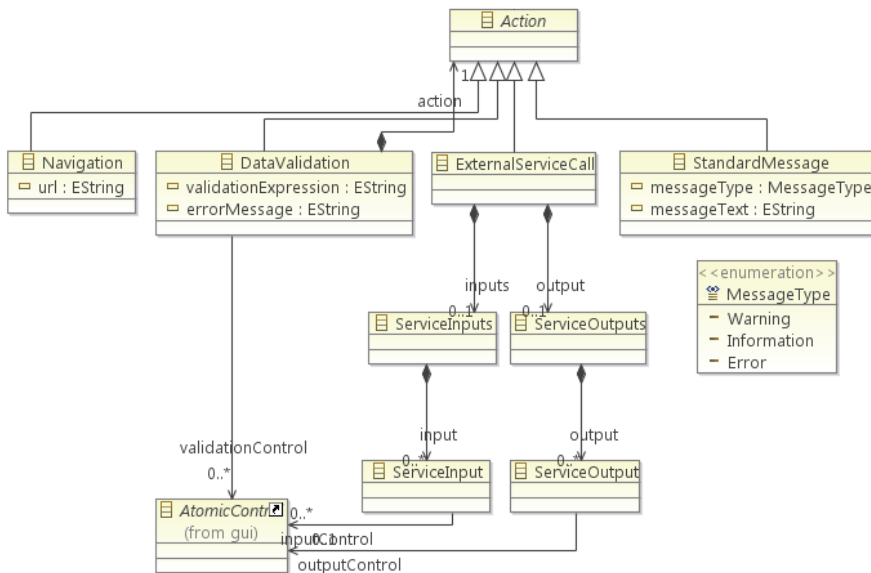


Figure 3.8: DSL metamodel for the *Action* activity.

igation action is used to navigate user from one page to another. It has one parameter, the url of the target page. DataValidation action is used to validate user input, for example, the email address input into the field. Therefore, this action has a reference to the AtomicControl to be validated. There have to be specified two parameters for this action. In the validationExpression parameter developer has to specify the regular expression to use for data validation. It was decided to use the regular expressions because it is easy way to check the correctness of the textual information such as user names, addresses, emails and more used in the Web forms. The second parameter is an errorMessage, which is message that will be shown to the user in case of invalid input. The

DataValidation action can include an activity which will be performed if the validation was successful. ExternalServiceCall is an action to call any external service, for example, Web service. It has a reference to an AtomicControl from which the data can be gathered and also to which the result could be output. The MessageCall is a standard action which is used to show the message to the user. The developer can define the message type and message text in according parameters of the MessageCall. There are defined three different kinds of message types: Warning, Error and Information. When the GUI view together with a dynamic behavior is defined the next step is to define a layout of the GUI. It is described in the next section.

Layout metamodel

Layout metamodel is used to define a layout of the GUI elements. On the CIM level in the GUI model the business consultant only defines the name and the description of the desired layout. The layout model is used to specify a concrete layout. The layout metamodel is shown in Figure 3.9

The Layout has only one attribute which is a name. There are two different layout types: the BasicLayout and the TableLayout. The BasicLayout defines a layout for simple elements. The TableLayout is used to define a table layout for the elements. The TableLayout is built of rows. The Row have two attributes which specify if it is a header or a footer row. If both of these parameters are set to false, it means that it is a body row. There can be many cells inside a row and each cell has a reference to the GUI control. The TableLayout cell can contain both the AtomicControl and the CompoundControl. As we can see from Figure 3.9 the LayoutType has an association with a ControlType from ASP metamodel which defines a concrete representation of the layout. The ASP metamodel is defined on the PSM abstraction level and therefore is described later.

Style metamodel

To define the style of elements such as size, text font, color and so on the designer needs to create a style metamodel. The style metamodel is fairly simple. Figure 3.10 depicts the metamodel.

Every GUI control can have a corresponding style specification which is defined On the CIM level by specifying its name and description. The style specification has an association with a concrete Style from a PIM level. The Style has a

name attribute and is composed of StyleRules, which have a property and a value attribute. The Style has a reference to StyleSpec of a GUI control. It is beneficial that the Style has a reference to the StyleSpec and not the other way around because several style specifications could be mapped to the same style. Therefore it is easier to define style once and reference the style specifications from it. The styles can be combined into the StyleCatalog. The styles for buttons, for example, can be grouped into button catalog. This makes it easier to group different styles. The collection of style catalogs can be reused for other applications.

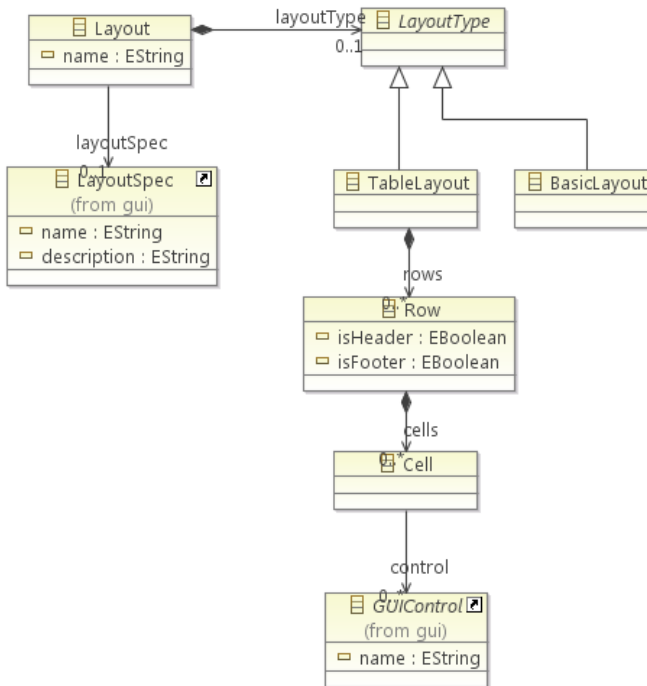


Figure 3.9: DSL metamodel for the *Layout*.

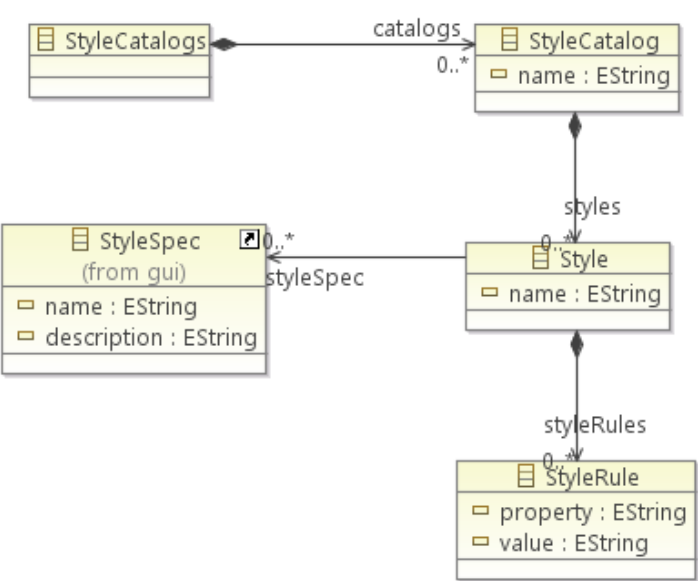


Figure 3.10: DSL metamodel for the *Style*.

3.5 DSL Metamodel for PSM

On the PSM level we created metamodel for ASP.NET specific implementation platform.

ASP metamodel

After the layout of GUI controls on the PIM level is defined, the developer has to specify the layout mapping to ASP concrete controls from the ASP metamodel. This metamodel specifies how the GUI element layout is implemented on the ASP.NET platform. The part of the metamodel is shown in Figure 3.11. The whole metamodel can be seen in Appendix A.

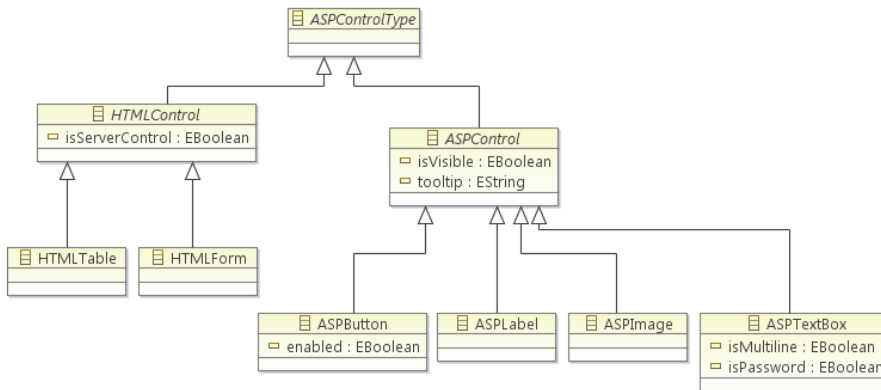


Figure 3.11: DSL metamodel for the *ASP*.

As it can be seen from Figure 3.11, we have divided `ASPControlType` into two subclasses `ASPControl` and `HTMLControl`. The first class represents the controls specific to ASP. As it is known, in ASP one can also define HTML controls which can be run on the server or on the client. These types of controls are represented by the `HTMLControl` class. The boolean attribute `isServerControl` is used to specify if this control will be run on the server or client.

The `ASPControl` class includes several ASP specific controls. Of course, the ASP metamodel can be very large but since we did not have the intention to build a full ASP metamodel we defined only a small part of it needed to create a running Web application example. As we can see we defined such

ASP controls as ASPText, ASPButton, ASPTextBox and several more. This metamodel elements define how the GUI control will look when the code will be generated.

3.6 Domain Specific Language Concrete syntax

In this section we will describe the concrete syntax of our domain-specific language on each level of abstraction.

3.7 CIM

On the CIM abstraction level we will use a graphical syntax to model the GUI prototypes. As an example we present a *Conveyor Operator* application login page. It consists of the logo of the company shown in the left corner. In the middle of the screen there are two fields with labels *username* and *password*. Under the fields there are two buttons: *Clear* and *Enter*. On the right side of the username field there is an information icon, which is used to show the information tooltip. Figure 3.12 shows the Login page.

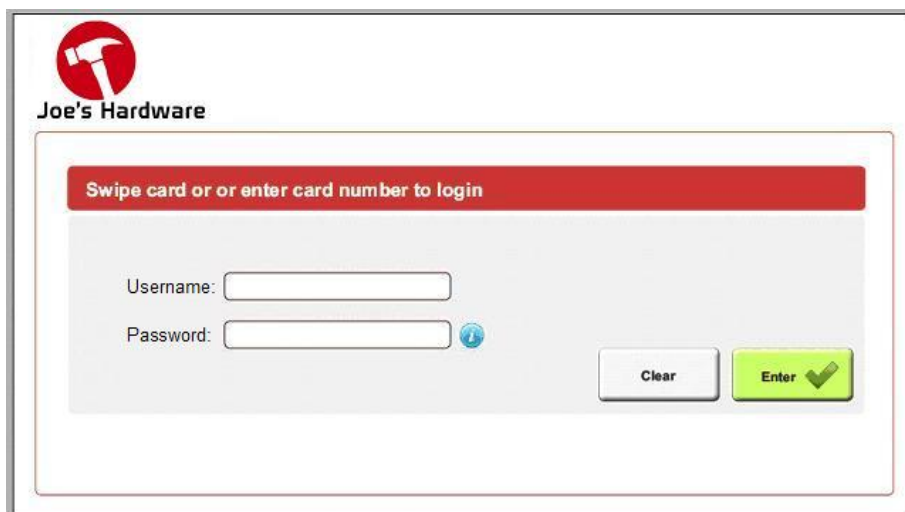


Figure 3.12: *Conveyor Operator* application Login page.

In our domain-specific language, the concrete syntax GUI controls are defined as rectangles with a control icon in the left top corner to be able to distinguish them. As described before we have defined only one CompoundControl which is a Container. A Container concrete representation is shown in Figure 3.13.



Figure 3.13: Concrete representation of a Container.

In the similar manner we have defined all other GUI controls. The figures representing the controls are all the same. To distinguish the different controls we created a different icon representing each of the controls. The icon is located in the left corner of the figure. The Button, Field, Text, Image, SingleSelect, MultiSelect as well as Style, Layout and Process specifications are defined with following icons:



Figure 3.14: Icon of the Button control in concrete syntax.



Figure 3.15: Icon of the Field control in concrete syntax.



Figure 3.16: Icon of the MultiSelect control in concrete syntax.



Figure 3.17: Icon of the Text control in concrete syntax.



Figure 3.18: Icon of the SingleSelect control in concrete syntax.

Figure 3.23 shows the Login page modeled using our modeling notation.



Figure 3.19: Icon of the Image control in concrete syntax.



Figure 3.20: Icon of the Layout specification in concrete syntax.



Figure 3.21: Icon of the Process specification in concrete syntax.



Figure 3.22: Icon of the Style specification in concrete syntax.

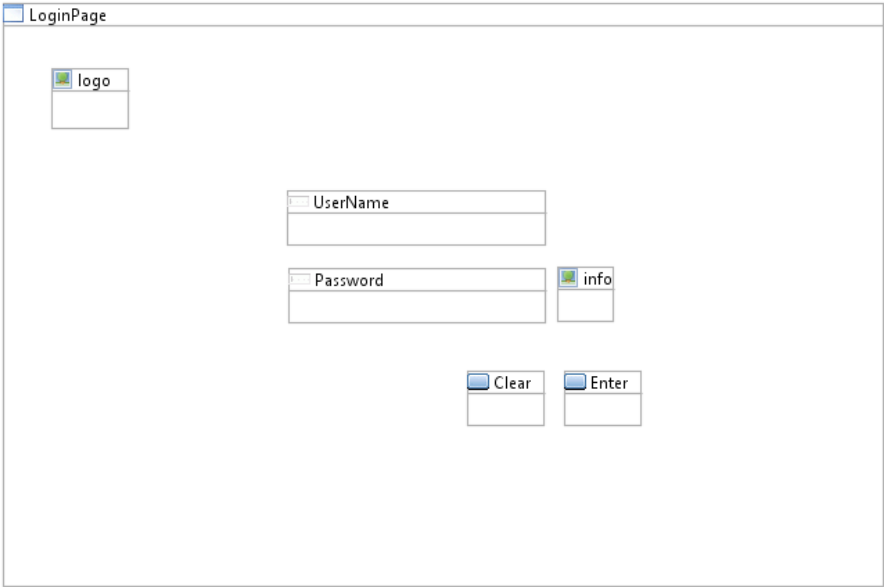


Figure 3.23: Concrete syntax of the Login page.

As it was described earlier, each of the GUI controls can have a process associated to it. In the CIM model we have to define the name, description and trigger for the process. In the generated software example a click on the Enter button triggers the Login process. The specification of the process can be defined in the properties of the ProcessSpec element as shown in the Figure 3.24.

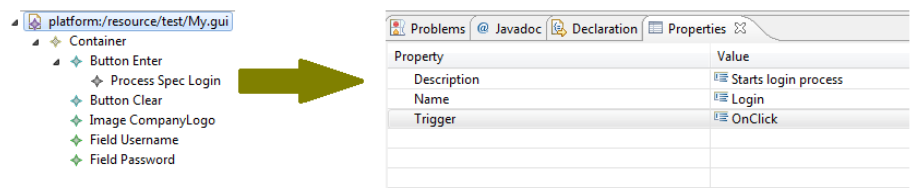


Figure 3.24: Properties of the ProcessSpec element.

3.8 PIM

On the CIM level, we defined our GUI model. We also specified that the Enter button triggers the Login process. On the CIM level we only defined the name, description and trigger of the process. On PIM level we have to define all details of the process. Unfortunately, time constraints did not allow us to implement a graphical editor for the process model. As mentioned earlier, we used a block strategy to model the process activities and in Figure 3.25 we show an example how the DataValidation action could look like in the concrete syntax.



Figure 3.25: DataValidation action concrete syntax.

The login process starts with a DataValidation of the username. For example, for the username the user has to use an email address, therefore we defined the

respective regular expression in the validationExpression attribute to validate the input. Also we defined the error message in case of the error. This can be done in the properties view as shown in Figure 3.26.

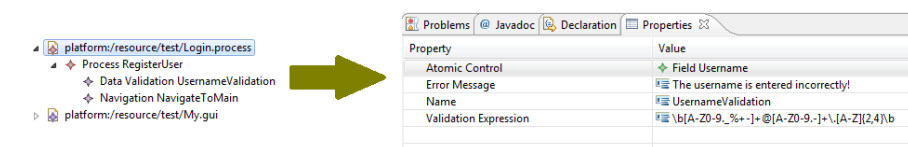


Figure 3.26: DataValidation action specification.

If the data is validated successfully, we redirect user to the Main Page, otherwise the DataValidation action defines that the error message will be shown to the user.

3.9 PSM

In the platform specific model we have to define a concrete ASP.NET implementation of the GUI control layout. For example, in the layout model we defined that the Field will have a BasicLayout. The layout should be further refined into a concrete implementation, we decided that a Field will be rendered as an APSTextBox control. In the properties view of the control we can add some specific information such as is the text box visible, is it a multiline text box etc.

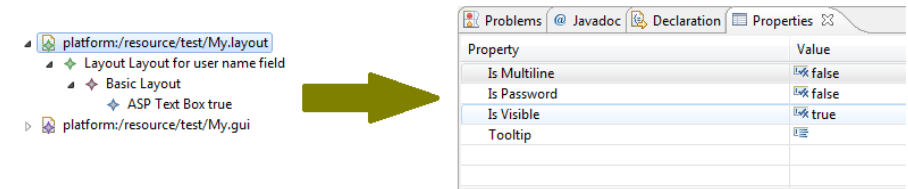


Figure 3.27: PSM ASP specific information definition.

3.10 Summary

In this chapter we have described the design principles and modeling notation of the domain specific language for model-based development of interactive Web applications. In the next chapter we will discuss the technologies used to implement the prototype tool for this language.

CHAPTER 4

Prototype Implementation

4.1 Overview

In the Chapter 4, we have described a Domain Specific Language for model-based development of interactive Web applications. In this chapter we will walk through the technologies used for the implementation of the Domain Specific Language. In order to implement the solution we used the technologies provided by Eclipse which is an open source, plug-in based software framework. In particular, the technologies we used were Eclipse Modeling Framework (EMF), Graphical Modeling Framework (GMF) and Model Development Tools (MDT) such as Eclipse OCL.

In our solution, it was decided to create a graphical editor for the GUI model development and several non-graphical editors for the development of process, style and layout models. To add additional semantic constraints to our modeling language we used the OCL. In order to generate the code of the application, several model-to-text transformations were defined as the XPand templates and code was automatically generated using XPand template engine.

4.2 DSL Editors Implementation

Eclipse Modeling Framework (EMF) is a powerful framework and code generation facility for building applications based on a structured model [53, 36]. Essentially, EMF contains three fundamental components: EMF.Core, EMF.Edit and EMF.Codegen. The EMF.Core consists of two elements: Ecore metamodel, which is used to define EMF models, and runtime support which provides model persistence using XMI, change notification, validation framework and a reflective API [55, 56]. Second, EMF.Edit supports displaying and editing instances of models in a basic tree-editor by providing content and label provider classes, property view and command classes that provide common services and operations. Third, EMF.Codegen provides a Java code generation of a complete editor for an Ecore model [55, 56].

The development process with EMF is depicted in Figure 4.1. Development using the EMF starts by creating an empty EMF project and defining the Ecore (.ecore) model. The framework allows defining a model using the Ecore tree-based editor and the Ecore class diagram editor. EMF also provides an opportunity to import models from Java interfaces, UML diagram or XML schema. After the Ecore model is created, the next step is to transform it to the generator model (.genmodel).

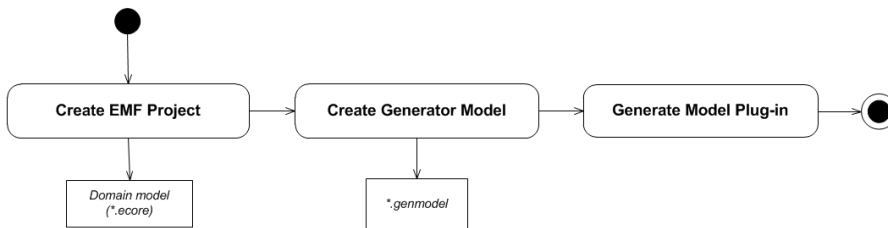


Figure 4.1: EMF development flow.

Basically, the generator model is a wrapper around the Ecore model which additionally provides access to all the data needed for generation. For example, it is used to specify where to put the generated code in the file system, which plugins to generate etc. From the generator model is used the model code and a basic tree-editor can be generated. The generated model code consists of Java interfaces and implementation classes for all the classes in the model, plus a factory and package (meta data) implementation class [56]. The editor code is split into two plugins: .edit and .editor. The .edit plug-in includes adapters

that provide a structured view and perform command-based editing of the model objects. The `.editor` plug-in code provides the UI for the editor and the wizard

4.3 DSL Implementation in EMF

In order to create our DSL, we used EMF built-in Ecore tree-based editor and the Ecore class diagram editor. As it was described in the previous chapter we created one DSL metamodel but divided it into several abstraction levels to separate different concerns of the Web application. For the same reason we also created different EMF editors to store instance models in separate files. We implemented the following editors:

- Editor for the GUI *.gui* (CIM level)
- Editor for the Process specification *.process* (PIM level)
- Editor for the Style specification *.style* (PIM level)
- Editor for the Layout specification *.layout* (PIM level)

We did not create a separate editor for the specification of the ASP.NET model. Conceptually, the ASP.NET model is on the PSM level, but in our design it was decided that it would be more convenient if developer could specify the concrete layout of the element when creating the layout specification model. Therefore, in our solution the concrete layout implementation information is specified using the layout specification model editor.

4.4 GMF Editors

As mentioned in the previous section using EMF one can automatically generate a basic tree-editor. However, in some cases the domain-specific language would benefit from having a graphical notation. To develop the graphical notation for the modeling language in Eclipse one could use the Graphical Modeling Framework (GMF) which provides a set of generative components and runtime infrastructures for developing graphical editors based on EMF [18].

To create a graphical editor we have to first create new GMF project and define the domain model (`.ecore`). Now we can construct the editor which is based on three models [54, 19]:

- Tooling Definition Model (.gmftool): This model is used to specify what elements will be available for diagramming on the editor canvas. Basically, it specifies palette tool definition which is UI control that displays a set of tools to be used in conjunction with the active diagram editor.
- Graphical Definition Model (.gmfgraph): This model is used to define the graphical representation of the domain model elements on the editor canvas.
- Mapping Definition Model (.gmfmap): This model is used to bind the elements from the Graphical Definition Model and the Tooling Definition Model to concrete classes of the domain model.

The Mapping Definition Model is a key model in GMF development since it will be used to produce the GMF generator model (.gmfgen). It is somewhat analogous to the EMF generator model and is used to set the properties for code generation. After the generation a new .diagram plugin is created in the workspace. The complete development process of the graphical editor using GMF is shown in Figure 4.2

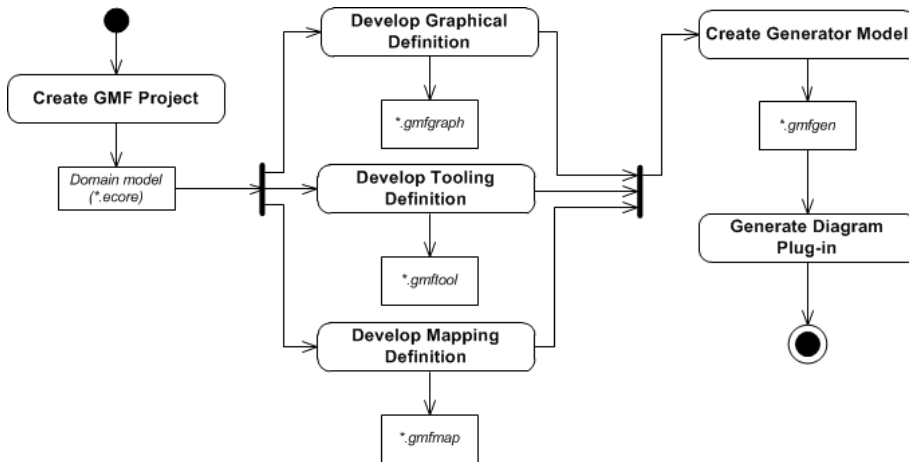


Figure 4.2: GMF development flow.

4.5 DSL Implementation in GMF

For our solution it was decided to create a graphical editor for creation of GUI models. The generated GMF editor is stored in `dk.dtu.mscproject.casetool.diagram` project. The graphical editor we have created is shown in the Figure 4.3

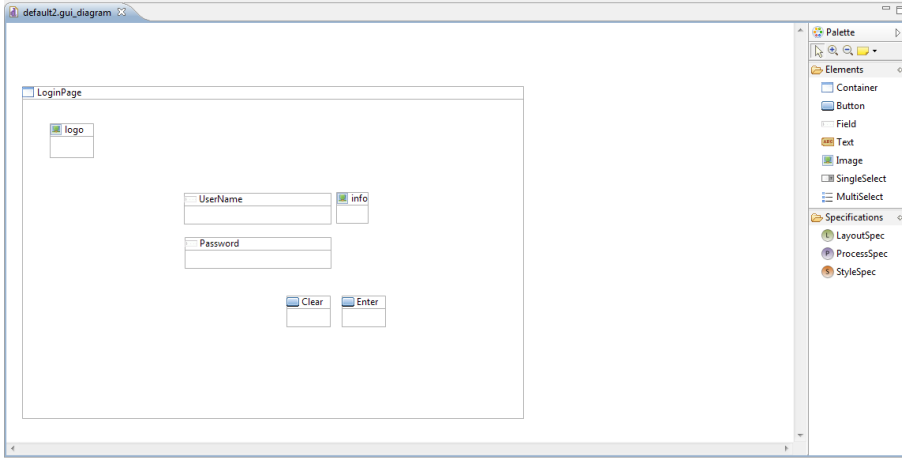


Figure 4.3: Graphical editor for GUI models.

As it can be seen in our graphical editor in the palette on the right we have created two tool groups: Elements and Specifications. Elements tool group contains a number of elements which we can use to create a GUI model. Specifications tool group contains different specifications we can define for the GUI elements such as process, style and layout specification.

In the Mapping Definition Model we had to make sure that the Container control has a correct mapping such that it would be able to contain other elements as well as other containers inside it. It is important because it affects how the resulting model tree will look.

4.6 Models Validation

Although EMF core provides basic validation support, there might be cases when additional constraints might be required. In EMF to ensure the model integrity we can use OCL to define additional semantic constraints. To apply OCL rules on model elements the OCL can be embedded in the models using annotations. Maintenance of these annotations is performed automatically by the OCLinEcore editor. The corresponding Java code is automatically generated by the EMF generator model.

In our solution we used OCL to define semantic constraints for the GUI model. In the GUI model it is important that ids of all elements are specified, because

they are used when generating the styles of the GUI elements and also in the application code when if we would like to reference the elements. To ensure this we specified the OCL rule which is shown in the Figure 4.4. Figure 4.4

```
context GUIControl inv:
  not (self._'id'.oclIsUndefined() and self._'id'<>'')
```

Figure 4.4: OCL rule for GUI control.

The first part of the rule ensures that model will not be validated if the id of any GUI control is null. The second part checks that the value is not an empty String. We had to add additional empty String constraint because oclIsUndefined operation will successfully validate the model in case of empty String but this must not be allowed. So now, if the business analyst will forget to specify the id of the elements, on the model validation he will get the following validation error as shown Figure 4.5

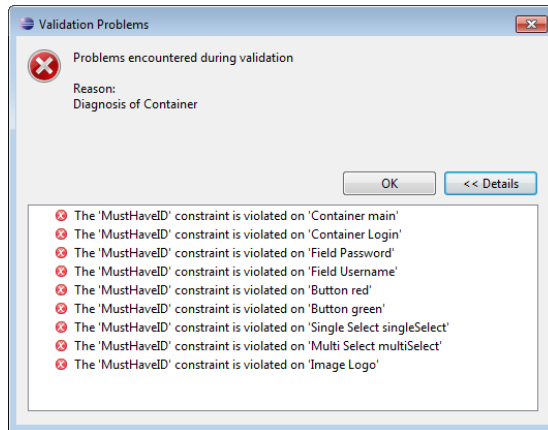


Figure 4.5: Model validation error.

Similar constraint were specified on other elements such as ProcessSpec which does not allow a trigger property to be empty as well as Decision and Loop conditions have to be specified. Also since Activity name is used as a name for method in code generation it has to be specified.

4.7 Developing Transformation

As described earlier to implement our solution we used the MDA incremental approach and developed one DSL metamodel which is divided into different levels of abstraction. These levels extend each other and therefore we don't need to create any model-to-model transformations. Instead, we have to create model-to-text transformation. In our solution we used the Xpand language to do it.

Xpand is a statically-typed template language that is used to control the output generation from the EMF models. The following example shows the transformation rule named StyleTemplate for style specification model element of type Style.

```

«REM» Run through each style «ENDREM»
«DEFINE StyleTemplate FOR Style->
«IF this.styleSpec.name.size>1»
  «FOREACH this.styleSpec.eContainer.typeSelect(GUIControl) AS c ITERATOR iter SEPARATOR ', '->
    #«c.id.toString()»
  «ENDFOREACH»
  «FOREACH styleRules AS rule»
    «EXPAND StyleRuleTemplate FOR rule->
  «ENDFOREACH»«"-»»
«ELSEIF this.styleSpec.name.size==1»
  #«this.styleSpec.eContainer.typeSelect(GUIControl).first().id.toString()»
  «FOREACH styleRules AS rule»
    «EXPAND StyleRuleTemplate FOR rule->
  «ENDFOREACH»
«ENDIF»
«ENDDEFINE»

```

Figure 4.6: Xpand transformation rule.

Using this rule we run through each style in the model and check to how many elements from the GUI model have this style applied. If it is more than one we run through all of them and print their names preceded by "#" sign and separated by comma. After that we call the StyleRuleTemplate which is used to generate each rule the style contains. If there is only one style we output its name and call StyleRuleTemplate which was explained before.

In order to run Xpand template we have to use the Modeling Workflow Engine (MWE) which is a declarative configurable generator engine. The engine provides a simple, XML based configuration language with which all kinds of generator workflows can be described. A generator workflow consists of a number of so called workflow components that are executed sequentially in a single JVM [20]. In the MWE file we have to specify which template to call and on which model instance. Once all the necessary information for the workflow is defined as well as the EMF model instance and the template are created the

generator can be run. This will produce a code file in the location specified in the MWE file.

In order to create the templates we used the Eclipse since it includes an editor where you can create the XPand templates. It also provides several nice features such as syntax coloring, error highlighting, navigation, refactoring and code completion.

In our solution we created following four templates and corresponding workflows:

- transform2gui_aspx.xpt (transform2gui_aspx.mwe)
- transform2gui_codebehind.xpt (transform2gui_codebehind.mwe)
- transform2gui_csharp.xpt (transform2gui_csharp.mwe)
- transform2gui_css.xpt (transform2gui_css .mwe)

The first three templates have to be applied on the GUI model (.gui). The first template will produce the .aspx file which specifies the visual content or presentation of the Web page. The second template will generate the code file that contains logic for interacting with the visual elements with the extension .aspx.cs. The third template is used to generate the business logic for the application and will generate a C# (.cs) file. The last template is applied on the style specification model (.style). It will generate the .css file containing all the styles for the application.

4.8 Limitations

To implement the solution using the MDA incremental approach, we, initially, created four separate .ecore models. Each model was created in the separate project and had a separate package. After that we created references between these models. Then we created a separate .genmodel for each .ecore model and generated corresponding editors which were described in Section 4.3.

In order to generate a Web page layout and process code we had to create a bidirectional reference between the GUI model and Layout model and between the GUI model and Process model. However, EMF allows creating bidirectional associations that cross .ecore packages only if all the code is generated together using one .genmodel. Therefore we had to combine the Layout and Process

models with GUI model and generate one common editor. So, in the final implemented prototype tool we have one GUI editor where user has to specify GUI, Layout and Process models and separate Style model editor. We tried to find a solution using the Xpand, however, it appeared that there is no way to access other model instance programmatically from the Xpand template. The template can be applied only on one model instance at a time. Therefore we modified the created templates and applied them on the GUI model but generated the code for each of the models in separate files as described in previous section.

4.9 Summary

A summary of the implemented features in the prototype can be described as following:

- The Domain Specific Language metamodel for all abstraction levels, such as CIM, PIM and PSM.
- Graphical editor for the CIM abstraction level, and non-graphical editors for the PIM and PSM levels of the DSL metamodel
- Validation of the models with additional constraints
- Transformation templates for all abstraction levels
- Generator workflows for all the templates
- Generation of the code for the ASP.NET

In this chapter we have described the implementation details of the prototype created to support the development process of interactive Web applications. In the next chapter we present the conclusions of the thesis.

Conclusions

In the last chapter we introduce the conclusions of the work presented in this thesis as well as directions for the future work.

5.1 Thesis summary

The objective of the thesis was to propose a solution to model requirements of interactive Web applications by using specific modeling notation to gather, specify and validate user requirements, as well as, by applying model transformations, to generate the code of the application. The hypothesis of the thesis was that usage of graphical user interface (GUI) models together with the models of the processes that are invoked by the user interaction on the GUI would improve the requirement specification process and reduce the amount of changes initiated by the customer in the later development phases.

There exists a variety of Web development languages and Web development platforms on the market and they continuously evolve. Because of these reasons, we decided to develop our modeling language using the Model Driven Engineering approach. This approach allowed us to describe the solution independently of the underlying programming language or specifics of execution platforms.

In particular, we used its well-known implementation framework called Model Driven Architecture (MDA).

There are few ways in applying the MDA approach. To develop our solution we used so-called incremental MDA approach. We chose this approach because it has several benefits in comparison to classical MDA approach. First of all, in this approach models on different levels extend each other which ensure the consistency of the models. Secondly, because models extend each other the model-to-model transformations are not required. Only the model-to-text transformations have to be developed. MDA incremental approach is divided into several levels of abstraction which allows Web application to be modeled from different viewpoints and by different people. These three levels of abstraction are: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). In our design, on the CIM level we model GUI prototypes in order to capture the requirements of the application, which can be done by a business consultant together with a customer. Additionally, in the GUI model business consultant can add process, layout and style specifications to different GUI elements. On the Platform Independent Model level the process specification is further refined into a process model by a developer. Similarly, layout and style information are further refined into layout and style models by a Web designer. On the last, PSM level, the developer has to specify the concrete representation of the GUI elements using ASP.NET model. When all models are specified the corresponding application code can be generated using model-to-text transformations.

As a proof of concept we developed a prototype tool implementing our modeling language. We implemented a graphical editor where business consultant is able to create graphical models of the GUI prototypes. To specify process, style and layout models we implemented non-graphical editors. For the model transformation we developed several model-to-text transformations using XPand template language. Using these templates the models are translated into the application code for the .NET platform. Our software was developed using technologies such as Eclipse Modeling Framework (EMF) and Graphical Modeling Framework (GMF) that are based on the Eclipse platform.

This project was made in collaboration with the Edora A/S company that gave us an opportunity to address real industry needs. Several times the consultants at the company experienced difficulties in capturing customer requirements. It was caused by the customer unfamiliarity with the UML diagrams such as use case diagrams and class diagrams. In the thesis, we proposed a way of capturing the requirements using the combination of the GUI and Process models. This improves the requirement elicitation process because the GUI prototype can be better understood by the customer and the Process model can capture the interactivity of the system. Since we used the model-driven development approach

to implement the solution, we were able to apply model transformations and automatically generate the code based on these models. In this way we could easily validate the requirements by running the application. We believe that the ability to simulate the behavior of the application in the early phase of the development process improves customers' understanding of his needs. Another advantage is that defined models are not disregarded after the requirement phase but can be used as the initial input for the for implementation of the complete application.

In order to evaluate our approach we used the Conveyor Operator application kindly provided by the Edora A/S company. Using our prototype it was possible to recreate the GUI designs and simple logic of the Conveyor Operator application. This allows us to conclude that principles of Model Driven Development can be applied for the elicitation, specification and validation of requirements of interactive Web applications.

5.2 Future work

Our approach covers only a part of the Web application development process and is mainly concerned with capturing and validating initial requirements of the application. Therefore, a number of improvements can be done to the approach and the prototype tool.

The main improvement on the Domain Specific Language would be to extend it with the opportunity to define a domain model. Partially, the domain model could be derived based on the information in the GUI and process models. Using the domain model it would be possible to automatically generate the underlying data model of the application.

In the prototype tool the usability of the model editors can be improved and additional code transformations can be developed. For example, it would be more intuitive to define the process models using a graphical editor instead of the tree editor. With regard to the code transformations they can be extended to provide broader coverage of the APS.NET platform. Also additional code transformations to cover other Web languages, such as JavaScript, PHP etc. could be defined.

APPENDIX A

Appendix A

In this chapter the complete metamodel of the Domain Specific Language is presented. Figure A.1 depicts the Style metamodel, Figure A.2 shows the GUI metamodel, Figure A.3 depicts the Process metamodel, Figure A.4 show the Layout metamodel without the ASP metamodel. Lastly, in A.5 the ASP metamodel is presented.

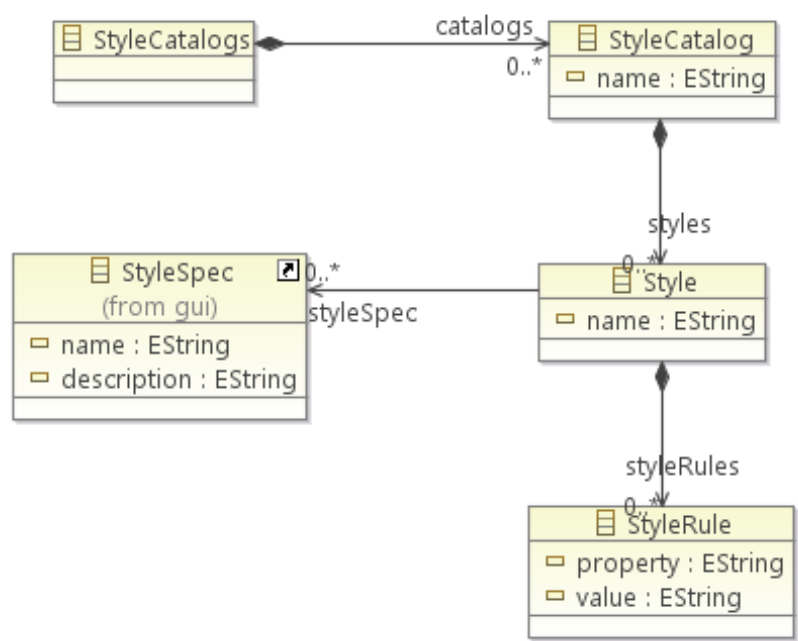


Figure A.1: Style metamodel

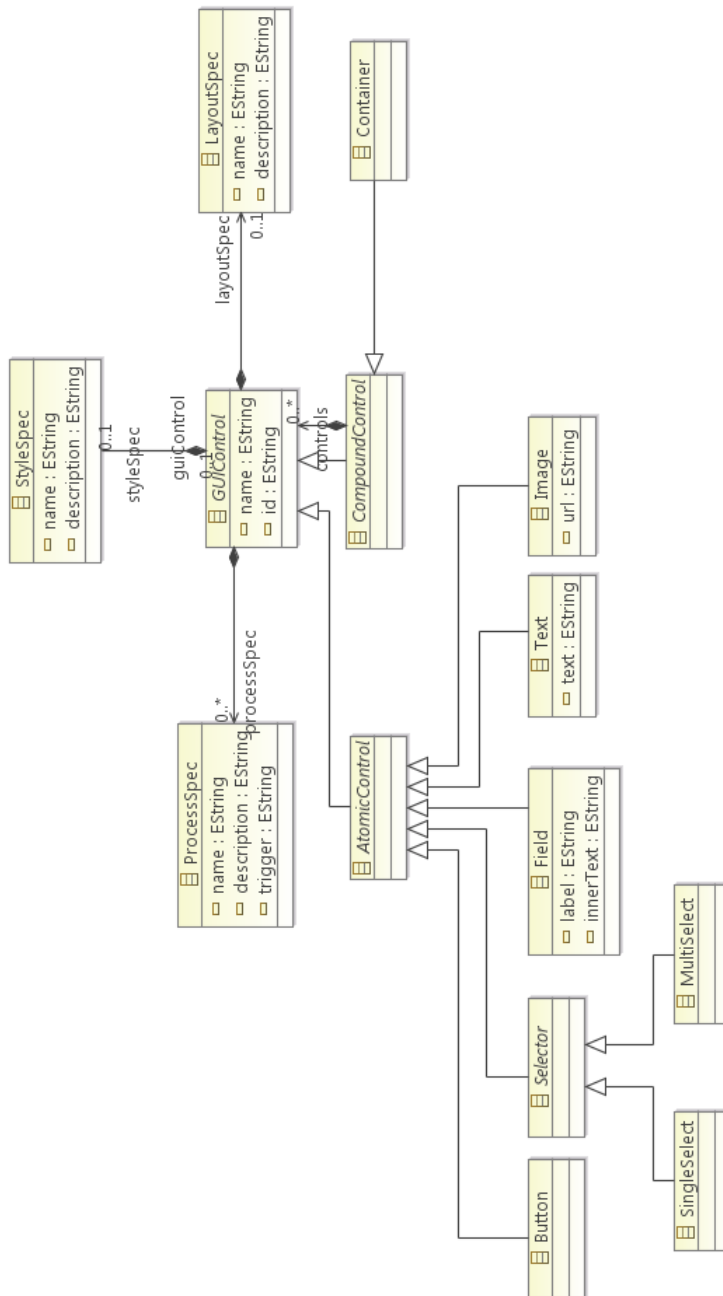


Figure A.2: GUI metamodel

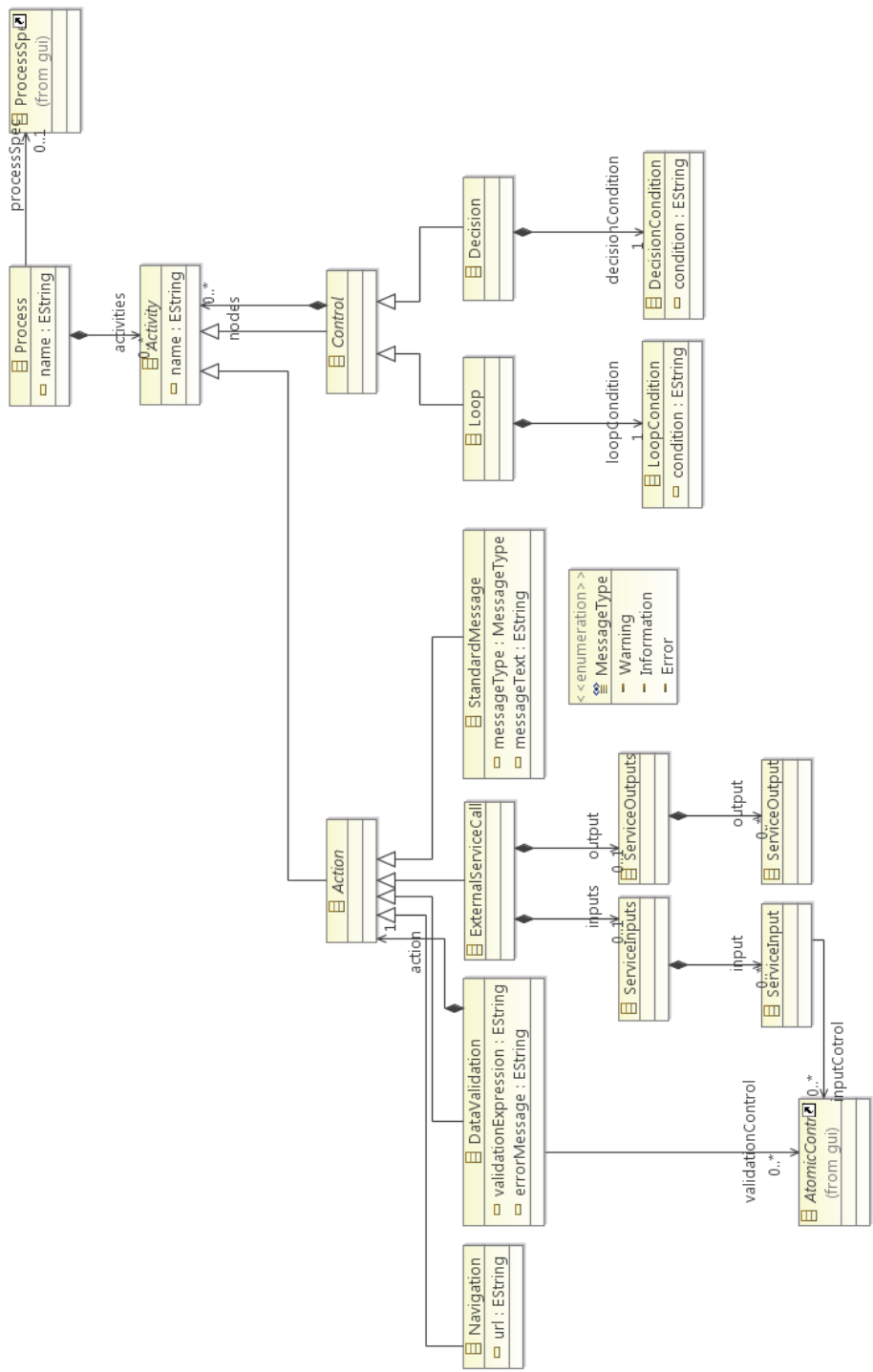


Figure A.3: Process metamodel

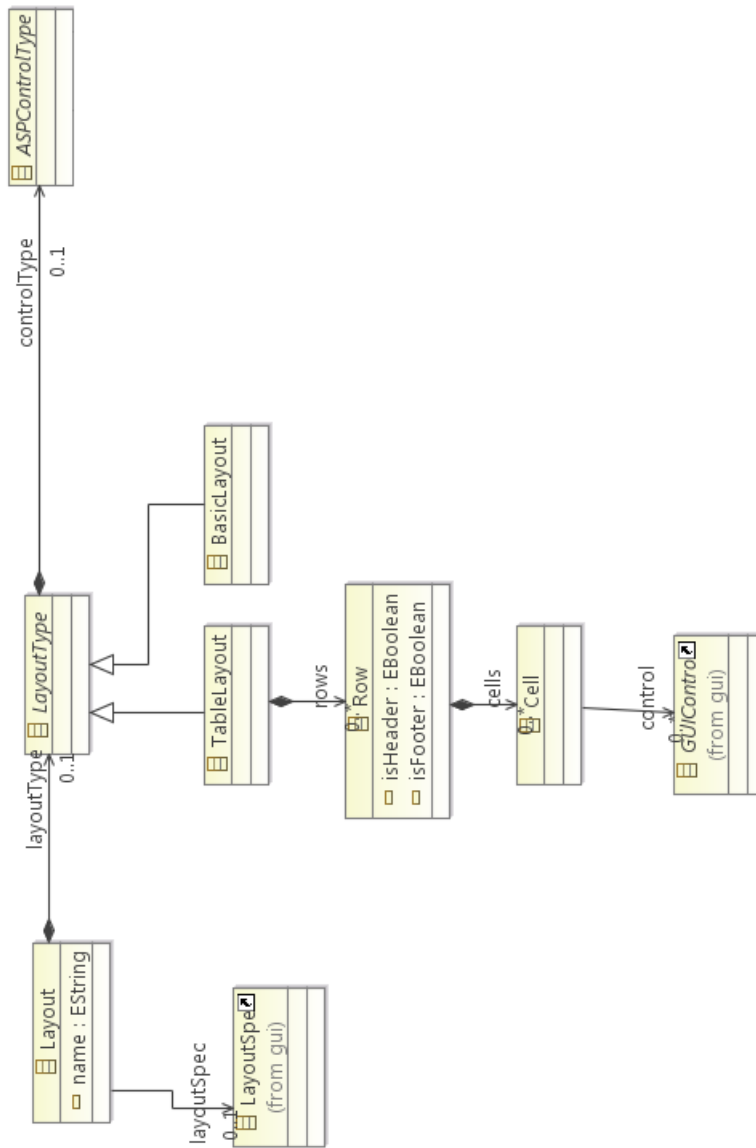


Figure A.4: Layout metamodel

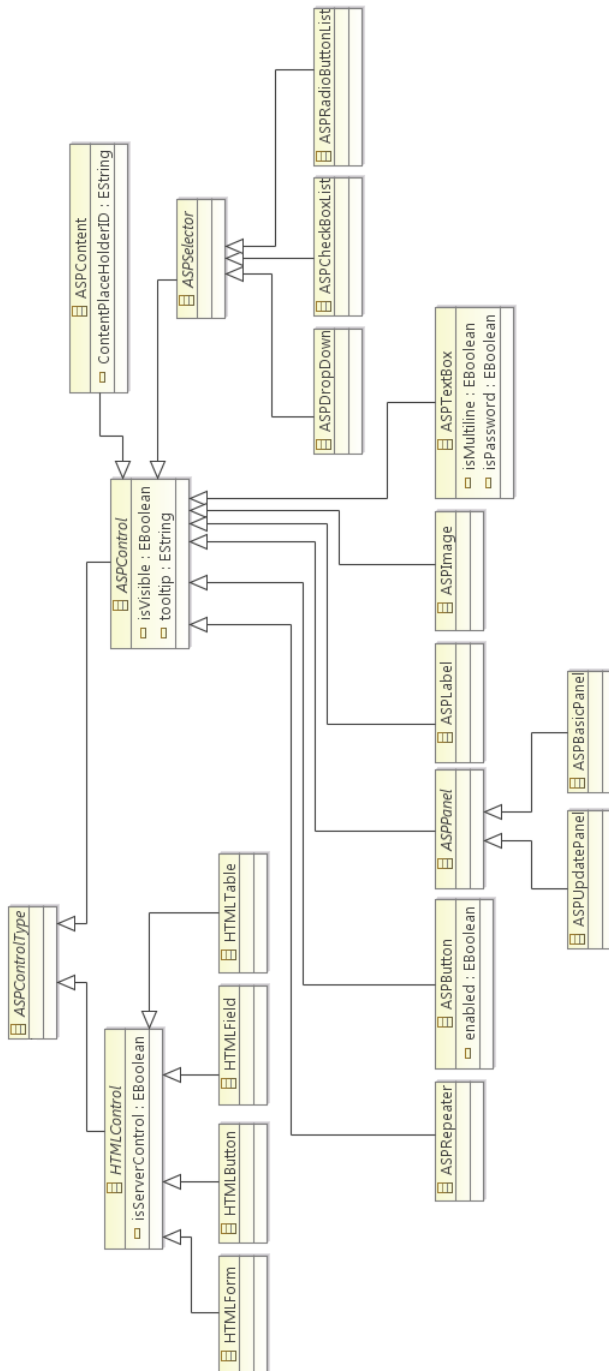


Figure A.5: Layout metamodel

Appendix B

In this chapter, we will explain how to install the prototype tool and demonstrate how to develop a DSL model for the Conveyor operator example Web application as well as transform it into the code for the ASP.NET platform. We will simplify the example and create only the Login, Main page and Production Line . We will create one process called Login and add it to Login button on the Login page and Navigation process from Main page to Production Line page.

B.1 Installation

1. Open the Eclipse IDE workbench.
2. Go to File → Import.
3. In the opened dialog select Existing Projects into Workspace.
4. Click the radio button next to Select archive file and click the Browse button to locate the project.
5. Find the archive file *dk.dtu.mscproject.casetool* on your hard disk. Click Open to select it.
6. Click Finish to perform the import.

7. Select Run → Run configurations... from the workbench menu bar.
8. Create new configuration and run it.
9. Import project *dk.dtu.mscproject.generator* into the runtime workbench the same way as casetool project previously.

B.2 User Guide

1. We start with the creation of three new GUI diagrams for Login, Main page and Production line page and save them in .gui_diagram files. It can be done by selecting New → Other → Examples → Gui diagram, as shown in Figure B.1

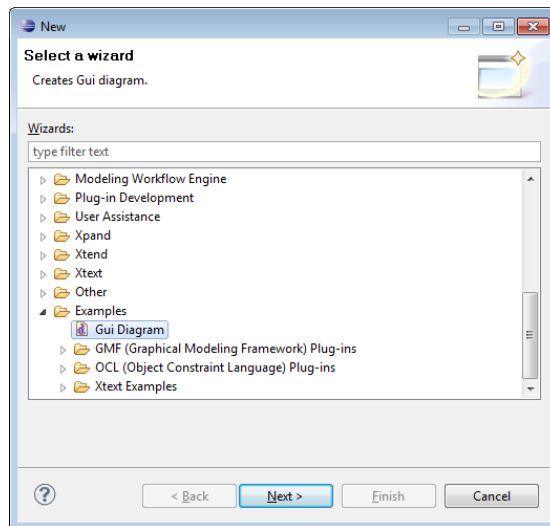


Figure B.1: Create a Login page GUI diagram file

2. We can create the Login page and Main page GUI prototypes by dragging necessary elements from the palette, as shown in Figure B.2.
3. When the GUI model is created we have to Validate it. From a pop-up-menu of the root model element in the GUI model select Validate, as shown in Figure B.3.

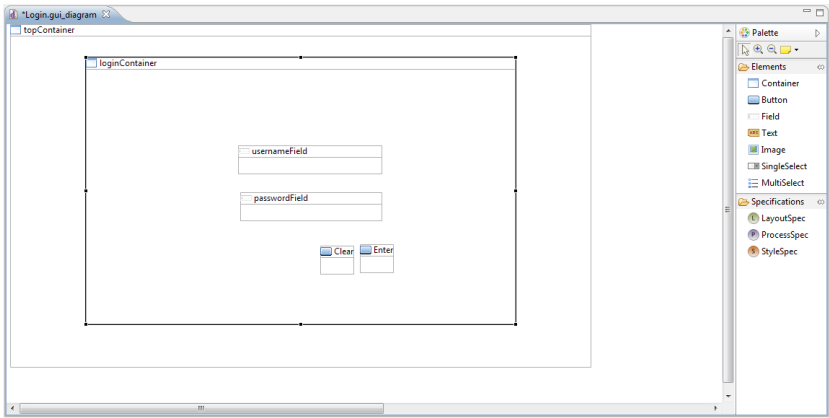


Figure B.2: Create a Login page diagram

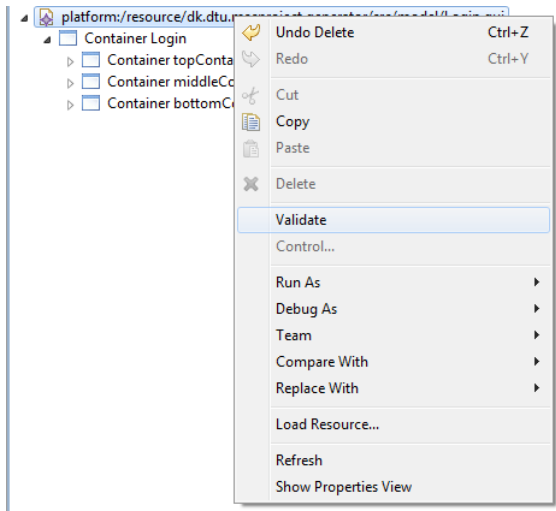


Figure B.3: Validation of the GUI model

4. We continue by creating the layout model using the tree editor. If we double-click on our Login.gui file, the Ecore tree editor will be opened. You can add layout information by right-clicking on the GUI control and adding LayoutSpec, then Layout, then you have to add the type of layout and, finally, add the concrete layout as it is shown in Figure B.4.
5. When the layout model is specified, we need to Validate the GUI model in the same way we did it in the step 3. Now we can continue extending

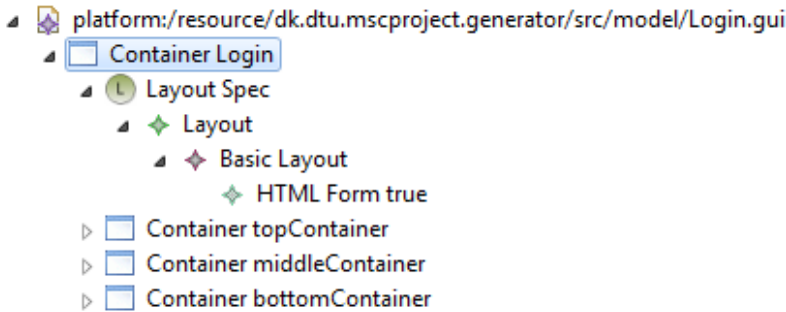


Figure B.4: Creating layout specification

our GUI model with Process model. We will define the Login process, which specifies that when the Enter button is clicked, if the input fields are filled in, the user will be redirected to the Main page, otherwise an error message will be shown. We add this process to the Enter button, as shown in the Figure B.5

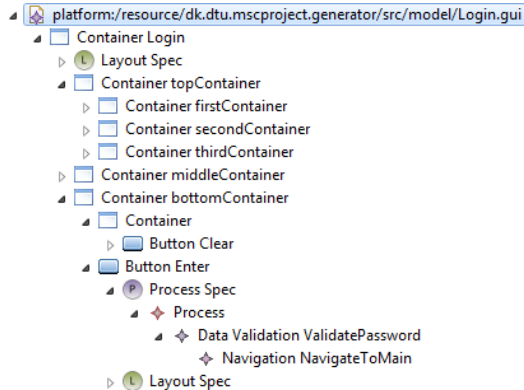


Figure B.5: Creating process specification

- When the Process model is created, we Validate the GUI model again as was described in step 3. The next step is to create a Style model. It can be done by selecting New → Other → Example EMF Model Creation Wizards → Style Model We will create a simple Style model which will define a green button style. The Style model is shown in the Figure B.6

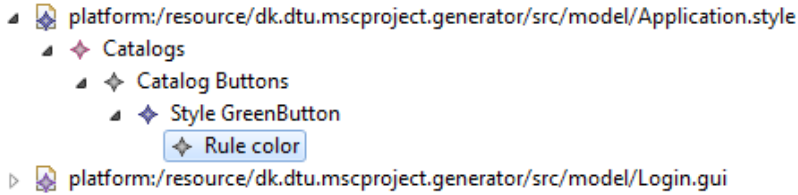


Figure B.6: Create a Style model

7. In order to associate a style with a GUI control we have to load the Login.gui file as a resource into the Style model and add references to the GUI elements. When the Styles are added, we have to Validate the model.

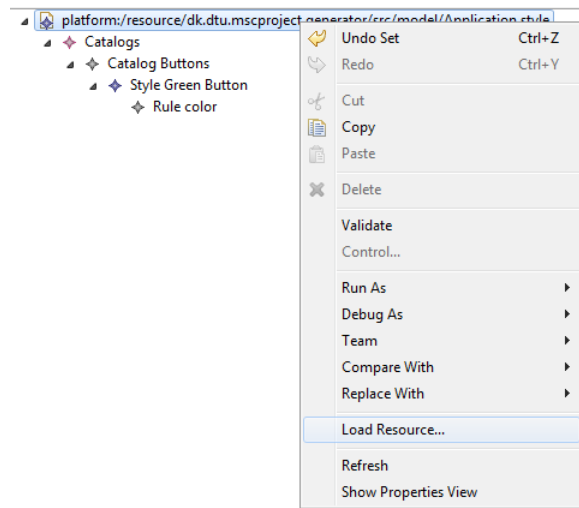


Figure B.7: Load the GUI model into the Style model

8. Now when all models are specified, we can generate the code. First, we will generate the .aspx file. To do it, we have to open the workflow.properties file and change the guiModelFile path to point to the GUI model we would like to generate code for. We should do the same also for the styleModelFile path which should point to the Style model.
9. Next we have to invoke a generator workflow. Press on the arrow near the Run button and choose the transform2ASPX.mwe as shown in Figure

B8. In the *src-gen* folder in the ASPX folder the generated .aspx file will appear.

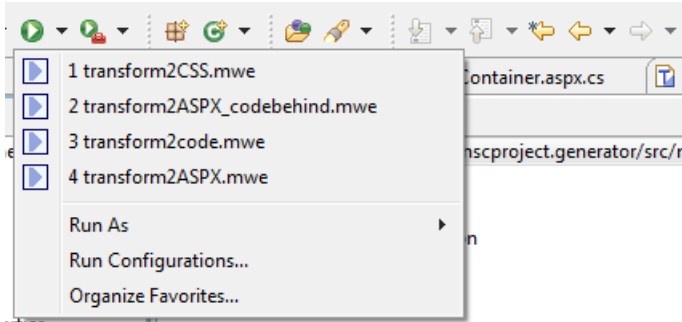


Figure B.8: generation of .aspx file

10. Now run the *transform2ASPX_codebehind.mwe* workflow. The code is generated into the corresponding folder under *src-gen* folder. The file with *aspx.cs* extension will be generated into the *ASPX_CS* folder. Figure B.9 shows the code generated for the *DataValidation* action.

```
protected void LoginButton_OnClick(object sender, EventArgs e)
{
    bool isValid = true;
    string[] ValidatePasswordTexts = {passwordField.Text, usernameField.Text};
    string sPattern = "^$";
    foreach (string s in ValidatePasswordTexts)
    {
        if (System.Text.RegularExpressions.Regex.IsMatch(s, sPattern))
        {
            showMessage("Validation Failed: Please input password and username!");
            isValid = false;
        }
    }
    if (isValid) {
        Response.Redirect("MainForm.aspx");
    }
}
```

Figure B.9: Contents of .aspx.cs file

10. Run the other workflows in the same manner. The C# code will be generated into the CSharp folder and style .css file will be generated into the CSS folder. Figure B.10 shows the *src-gen* folder after the code generation.

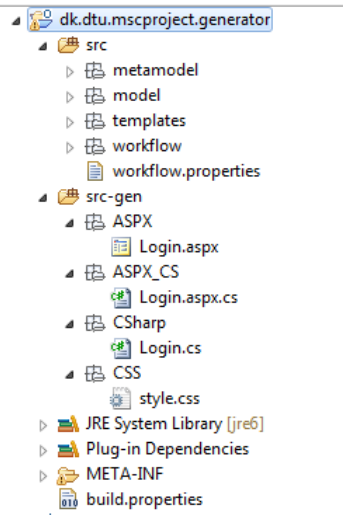


Figure B.10: Generated files in the Eclipse

12. When application files are generated we have to test it. In order to do it we will open them using Visual Studio. We will create empty ASP project and put our files there as shown in the Figure B.11.

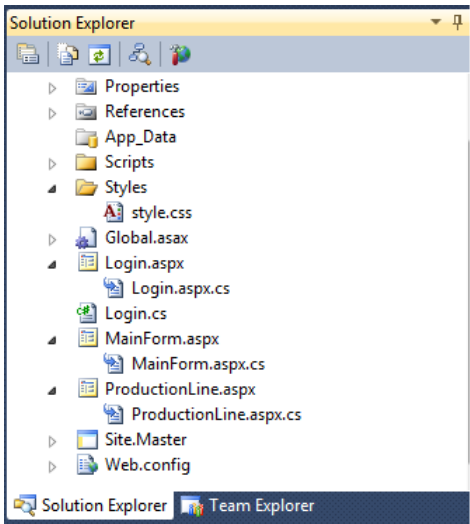
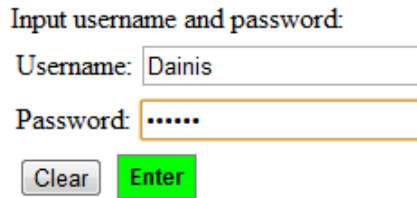


Figure B.11: Generated files in Visual Studio project

13. We can run the project by pressing F5. ASP has a built-in Web Server which easily allows us to test the application. The Login page is shown in the Figure B.12. The generated Main Page is shown in Figure B.13. Lastly, the generated Production Line page is shown in the Figure B.14

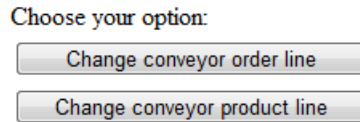


Input username and password:

Username:

Password:

Figure B.12: Generated Login page opened in the browser



Choose your option:

Figure B.13: Generated Main page opened in the browser

Change Production Line

Filter

NO	Name	
100	Production Line 100 name	<input type="button" value="Choose"/>
101	Production Line 101 name	<input type="button" value="Choose"/>
102	Production Line 102 name	<input type="button" value="Choose"/>

Figure B.14: Generated Production Line page opened in the browser

Bibliography

- [1] Maurice de Kunder. *Size of World Wide Web*. Available at: <http://www.worldwidewebsite.com>, accessed January 2012.
- [2] Miniwatts Marketing Group. *Internet Usage Statistics*. Available at: <http://www.internetworldstats.com/stats.htm>, accessed January 2012
- [3] Busch M., Koch N. *Rich Internet Applications. State-of-the-Art*. Technical Report, Ludwig-Maximilians-Universität München, 2009.
- [4] Wright J., Dietrich J. *Survey of Existing Languages to Model Interactive Web Applications*. In Proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling (APCCM 2008), Wollongong, NSW, Australia, 2008.
- [5] Lowe D., Eklund J. *Client Needs and the Design Process in Web Projects*. Journal of Web Engineering, 1(1), published electronically on CD-ROM, 2002.
- [6] Murugesan S. *Web Application Development: Challenges And The Role Of Web Engineering*. In Web Engineering: Modelling and Implementing Web Applications, Springer London, pp. 7-32, 2008.
- [7] Barry C., Lang M. *A Survey of Multimedia and Web Development Techniques and Methodology Usage*. IEEE MultiMedia, 8(2), pp. 52-60, 2001.
- [8] Koch N., Knapp A., Zhang G., Baumeister H. *UML-based Web Engineering: An Approach based on Standards*. In Web Engineering and Web Applications Design Methods, Human-Computer Interaction Series, vol. 12, chapter 7. Springer, 2007.

- [9] Ceri S., Fraternali P., Bongio, A. *Web Modeling Language (WebML): a Modeling Language for Designing Web Sites*. 9th International WWW Conference, Amsterdam, pp. 137-157, 2000.
- [10] Pastor, O., Fons, J., Pelechano, V., Abrahao, S. *Conceptual modelling of Web applications: the OOWS approach*. Theory and Practice of Metrics and Measurement for Web Development, Mendes E., Springer, pp. 277-302, 2005.
- [11] Gómez J., Cachero C. *OO-H Method: extending UML to model web interfaces*. Information modeling for internet applications, Idea Group Publishing, 2003.
- [12] A. Vallecillo, N. Koch, C. Cachero, S. Comai, P. Fraternali, I. Garrigos, J. Gomez, G. Kappel, A. Knapp, M. Matera, S. Melia, N. Moreno, B. Proll, T. Reiter, W. Retschitzegger, J.E. Rivera, W. Schwinger, M. Wimmer, G. Zhang. *MDWEnet: A Practical Approach to Achieving Interoperability of Model-Driven Web Engineering Methods*. In Proceedings of Third International Workshop Model-Driven Web Eng., pp. 246-254, 2007.
- [13] Escalona M.J., Koch N. *Requirements Engineering for Web Applications: A Comparative Study*, *Journal on Web Engineering*. 2(3), pp. 193-212, 2004.
- [14] Escalona M. J., Aragon G., *NDT. A model-driven approach for web requirements*. IEEE Trans Softw Eng, 34 (3), pp. 377-390, 2008.
- [15] Schwabe D., Rossi G., Barbosa S. *Systematic Hypermedia Design with OOHDm*. 7th ACM International Conference on Hypertext, ACM Press, Washington, pp. 116-128, 1996.
- [16] Lowe D. *Web Requirements: An Overview*. *Requirements Engineering Journal*. 8(2), pp. 102-113, Springer, 2003.
- [17] Lee H., Lee C., Yoo C. *A scenario-based object-oriented hypermedia design methodology*. *Information & Management*. 36, pp. 121-138, 1999.
- [18] Eclipse Foundation. *Graphical Modeling Project (GMP)*. Available at: <http://www.eclipse.org/modeling/gmp/>, Accessed: March 2012.
- [19] IBM. *Tutorial: Configuring and Extending the Diagram Palette*. Available at: <http://publib.boulder.ibm.com/infocenter/rsmhelp/v7r0m0/index.jsp?topic=/org.eclipse.gmf.doc/tutorials/diagram/paletteConfigurationTutorial.html>, Accessed: March 2012.
- [20] Efftinge S., Kadura C. *OpenArchitectureWare 4.1: Xpand Language Reference*. Available at: http://www.openarchitectureware.org/pub/documentation/4.1/r20_xPandReference.pdf, Accessed: March 2012

- [21] Cormode G., Krishnamurthy B. *Key differences between Web1.0 and Web2.0*. First Monday, 13(6), June 2008.
- [22] Duhl J. *Rich Internet Applications*. White Paper, IDC, November, 2003.
- [23] Loosey C. *Rich Internet Applications: Design, Measurement, and Management Challenges*. White Paper, Keynote, 2006.
- [24] IBM Corporation. *IBM: Putting the Power of Web 2.0 into Practice How Rich Internet Applications can deliver Tangible Business Benefits*. Available at: <ftp://ftp.software.ibm.com/software/lotus/lotusweb/product/expeditor/LOW14003-USEN-00.pdf>, Cambridge, USA, 2008.
- [25] Deb B., Bannur S.G., Bharti S. *Rich Internet Applications (RIA) Opportunities and Challenges for Enterprises*. Available at: <http://www.infy.co.uk/technology/rich-internet-applications.pdf>, White Paper, InfoSys, 2007.
- [26] Garrett J. J. *Ajax: A New Approach to Web Applications*. Available at: <http://www.adaptivepath.com/publications/essays/archives/000385.php/>, Technical report, 2005.
- [27] Boydens J., Steegmans E. *Model driven architecture: The next abstraction level in programming*. First European Conference on the Use of Modern Information and Communication Technologies (ECUMICT), pp. 97-104, Nevelland, 2004.
- [28] Schmidt C. D. *Model-Driven Engineering*. IEEE Computer, 39(2), February 2006.
- [29] Sommerville I. *Software Engineering*. Addison-Wesley, 9th edition, 2011.
- [30] Tveit M.S. *A Meta-Model-Based Approach for Specification of Graphical Representations*. Available at: <http://dblp.uni-trier.de/db/journals/eceasst/eceasst18.html#Tveit09>, ECEASST, 18.s., 2009.
- [31] Kleppe A. *MDA Explained, The Model Driven Architecture: Practise and Promise*. ISBN 032119442X, Addison-Wesley, 2003.
- [32] OMG. *MDA guide version 1.0.1*. Available at: <http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>, Accessed: March 2011.
- [33] Bézin J., Gerbe O. *Towards a Precise Definition of the OMG/MDA Framework*. In ASE 01: Proceedings of the 16th IEEE international conference on Automated software engineering, Washington, DC, USA, IEEE Computer Society, 2001.

- [34] OMG. *Meta Object Facility (MOF) Core Specification*. Version 2.0, OMG Document formal/06-01-01, Available at: <http://www.omg.org/cgi-bin/doc?ptc/03-10-04.pdf> Accessed: 2011, March.
- [35] Liddle S. W. *Model-Driven Software Development*. June 2010.
- [36] ATHENA. *Model-Driven Interoperability (MDI) Framework*. Available at: <http://www.modelbased.net/mdi/mda/mda.html>, Accessed: March 2011.
- [37] Deursen A., Klint P., Visser J. *Domain-Specific Languages: An Annotated Bibliography*. ACM SIGPLAN Notices, 35(6), pp. 26-36, 2000.
- [38] Bezivin J. *From Object Composition to Model Transformation with the MDA*. In Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), IEEE Computer Society, Washington, DC, USA, TOOLS '01.
- [39] Bezivin, J., Ploquin, N. *Tooling the MDA framework: a new software maintenance and evolution scheme proposal*. Application Development Trends, 2001.
- [40] Stephen J. Mellor, Kendall S., Uhl A., Weise D. *MDA Distilled*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.
- [41] Kindler E. *Model-Based Software Engineering and Process-Aware Information Systems*. Transactions on Petri Nets and Other Models of Concurrency, 5460:2745, 2009.
- [42] Seidewitz E. *What Models Mean*. IEEE Software, vol. 20, no. 5, pp. 26-32, September/October 2003.
- [43] OMG. *UML 2.0 Superstructure*. Available at: <http://www.omg.org/spec/UML/2.0/>, Accessed: March 2011.
- [44] Hennicker R., Baumeister H., Knapp A., Wirsing M. *Specifying Component Invariants with OCL*. GI Jahrestagung (1), pp. 600-607, 2001.
- [45] Djuric D., Gasevic D., Devedzic V. *The Tao of Modeling Spaces*. Journal of Object Technology 5(8), pp. 125-147, 2006.
- [46] Aguilar J.A. , Garrigós I., Mazón J., Trujillo J. *Web Engineering Approaches for Requirement Analysis - A Systematic Literature Review*. In Proc. WEBIST (1), pp. 187-190, 2010.
- [47] Microsoft. *Windows User Experience Interaction Guidelines*. Available at: <http://msdn.microsoft.com/en-us/library/windows/desktop/aa511258.aspx>, Accessed: March 2012.

- [48] Apple. *User Experience Guidelines*. Available at: http://developer.apple.com/library/mac/#documentation/UserExperience/Conceptual/AppleHIGuidelines/UEGuidelines/UEGuidelines.html#//apple_ref/doc/uid/TP40002720-TPXREF101, Accessed: March 2012.
- [49] The GNOME Project. *GNOME Human Interface Guidelines 2.2.2*. Available at: <http://developer.gnome.org/hig-book/3.0/>, Accessed: March 2012.
- [50] W3C. *HTML5*. Available at: <http://dev.w3.org/html5/spec/Overview.html>, Accessed: March 2012.
- [51] Gamma E., Helm R., Johnson R., Vlissides J. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, 1995.
- [52] Tsyukh U. *A Formal Modeling Notation for the Requirements of Workflow Systems*. Master's thesis, Technical University of Denmark, August 2010.
- [53] Steinberg D., Budinsky F., Paternostro M., Merks E. *EMF: Eclipse Modeling Framework 2.0*. 2nd edition, Addison-Wesley Professional, 2009.
- [54] Gronback R.C. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, Addison-Wesley Professional, 2009.
- [55] Dingel J. *Eclipse Modeling Framework*. Available at: <http://miloud.webs.com/emf.htm>, Accessed: March 2012.
- [56] Eclipse Foundation. *Eclipse Modeling Framework Project (EMF)*. Available at: <http://www.eclipse.org/modeling/emf/?project=emf>, Accessed: March 2012.
- [57] Rossi, G., Schwabe, D. *Web Engineering: Modelling and Implementing Web Applications*. Chap. Modeling and Implementing Web Applications with Oohdm, Human-Computer Interaction Series, pp. 109-155, 2008.
- [58] Preciado J.C., Linaje, M., Sánchez. F. *Necessity of methodologies to model Rich Internet Applications.*, In Proceedings of the Seventh IEEE International Symposium on Web Site Evolution (WSE '05). IEEE Computer Society, Washington, DC, USA, pp. 7-13.
- [59] Costagliola G., Ferrucci F., Francese R. *Web Engineering: Models and Methodologies for the Design of Hypermedia Applications.*, Handbook of Software Engineering & Knowledge Engineering, vol. 2, Emerging Technologies, pp. 181-199, World Scientific, 2002.
- [60] Schwabe D., Rossi G., Barbosa S. D. J. *Systematic hypermedia application design with OOHDm*. Proceedings of the the seventh ACM conference on Hypertext, 1996.

- [61] Cachero C., Gómez J., Párraga A., Pastor O. *Conference Review System: A Case of Study.*, in Proceedings of the 1st International Workshop on Web Oriented Software Technology (IWWOST '01), 2001.
- [62] Wright J., Dietrich J. *Survey of Existing Languages to Model Interactive Web Applications.*, In Proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling (APCCM 2008), Wollongong, NSW, Australia, 2008.
- [63] Schwinger W., Koch N. *Web Engineering: Systematic Development of Web Applications.* Chap. Modeling Web applications, pp. 39-64, 2006
- [64] Koch N., Knapp A., Zhang G., Baumeister H. *Web Engineering: Modelling and Implementing Web Applications.* Chap. Uml-Based Web Engineering, pp. 157-191, Human-Computer Interaction Series, London, 2008.
- [65] Koch, N., Kraus, A., Cachero, C., Meliá, S. *Modeling Web Business Processes with OO-H and UWE.* 3rd International Workshop on Web Oriented Software Technology (IWWOST03), Oviedo, Spain, 2003.