UNIVERSITY OF OSLO
Department of Informatics

MULTIDEV

- a model based framework for rapid development of multi-platform clients in a distributed environment

Gunnar Lunde

Cand Scient Thesis

February 2007

Abstract

This thesis presents a framework for generating clients to computer systems that are based on COMDEF. COMDEF was an early attempt at Modeling Based Development. While working on this thesis, the development of COMDEF has stopped. However, there are many new initiatives that provide similar frameworks, such as MDA from OMG and DSL from Microsoft.

The framework presented in this thesis is called Multidev and is a model-based framework for developing multi platform clients in a distributed environment. The distributed environment in this context is systems that are developed with COMDEF. Multidev tries to utilize the information in the COMDEF UML model of a system to generate as much of the client as possible, with as little intervention by the developer as possible. The Multidev framework includes a high level tool with a graphical modeling environment for user interfaces that lift the abstraction level from implementation to modeling of the user interfaces.

Multidev was developed as an add-on to the COMDEF framework, but the focus of the Multidev framework is independent of the actual framework used. Therefore, with the experience from COMDEF as a base, it should be possible to transfer the Multidev framework to similar frameworks, such as the OMG MDA and/or DSL from Microsoft.

Acknowledgments

This thesis is submitted in partial fulfillment of the *Cand. Scient.* degree in informatics at the Department of informatics, University of Oslo (UiO). It is written partly at SINTEFs premises in Oslo as well as at the University Of Oslo and in my home in Bærum.

I would like to thank my supervisor Dr. Arne Jørgen Berre for his guidance and help throughout the work of this thesis.

Special thanks also to all of my family for help and to my absolutely closest for allowing me time to work on this thesis.

Bærum, February 27, 2007 Gunnar Lunde

Table of Contents

TABLE OF CONTENTS **7**

LIST OF FIGURES **11**

CHAPTER 1 INTRODUCTION **13**

1.1 BACKGROUND 13
1.1.1 Research in graphical user interfaces *13*
1.1.2 Model- and component based software development *15*
1.2 COMDEF 15
1.3 CONTEXT OF THESIS 17
1.4 STRUCTURE OF THESIS 17
1.5 SUMMARY 18

CHAPTER 2 CASE STUDY **19**

2.1 THE CAR RENTAL MODEL ELEMENTS 20
2.1.1 Car entity *20*
2.1.2 Reservation Entity *20*
2.1.3 Customer entity *21*
2.1.4 Services *21*
2.1.5 User service *21*
2.1.6 Events *21*
2.2 USE CASES 21
2.2.1 Register new customer *22*
2.2.2 Get customer *22*
2.2.3 Edit customer *22*
2.2.4 Make reservation *23*
2.2.5 Get reservation *23*
2.2.6 Edit reservation *23*
2.2.7 Check out car *23*
2.2.8 Check in car *23*
2.3 GENERAL CLIENT REQUIREMENTS 24
2.4 PROBLEM AREAS IDENTIFIED 24
2.5 SUMMARY 24

CHAPTER 3 REQUIREMENTS **25**

3.1 PRODUCTIVITY 25
3.1.1 Code generation *25*
3.1.2 Independency *26*
3.1.3 General *26*
3.2 MODELING CONCEPTS 26
3.2.1 General *27*
3.2.2 Architecture *27*
3.3 MULTIPLE PLATFORM SUPPORT 27
3.4 EVOLUTION 28
3.5 EVALUATION CRITERIA 28
3.6 SUMMARY 28

CHAPTER 4 EVALUATION OF EXISTING WORK **29**

4.1 GENERA GENOVA OVERVIEW 29
4.1.1 Genova work protocol *29*

4.1.2 Implementing the case using Genova ... *32*
4.1.3 Evaluation of Genova ... *33*
4.1.4 Modeling concepts ... *34*
4.1.5 Multi platform support ... *34*
4.1.6 Evolution ... *34*
4.2 MOBI-D OVERVIEW ... 34
4.2.1 Mobi-d work protocol ... *35*
4.2.2 Implementing the case with Mobi-d ... *36*
4.2.3 Evaluation of Mobi-d ... *36*
4.3 INTERFACE BUILDERS OVERVIEW ... 38
4.3.1 Visual Café work protocol ... *38*
4.3.2 Implementing the case using Visual Café ... *38*
4.3.3 Evaluation of Visual Café ... *39*
4.3.4 Modeling concepts ... *39*
4.3.5 Multi platform support ... *40*
4.3.6 Evolution ... *40*
4.4 SUMMARY ... 40

CHAPTER 5 MULTIDEV'S TECHNOLOGY RELATIONSHIPS ... 41

5.1 COMDEF ... 41
5.1.1 The user service ... *42*
5.1.2 The services ... *42*
5.1.3 The entities ... *43*
5.1.4 Events ... *43*
5.1.5 Mapping ... *43*
5.1.6 Post COMDEF ... *43*
5.2 XML ... 43
5.3 XMI ... 43
5.4 XSLT ... 44
5.5 UIML ... 44
5.6 JAVA ... 45
5.7 UML ... 45
5.8 SUMMARY ... 46

CHAPTER 6 MULTIDEV ... 47

6.1 LIMITATIONS AND ADVANTAGES ... 47
6.2 MULTIDEV OVERVIEW ... 47
6.2.1 The Multidev process ... *49*
6.3 ARCHITECTURE ... 50
6.4 XMI AND CML ... 52
6.5 UML MODELING ... 52
6.6 EXTENDING THE UML METAMODEL ... 53
6.7 EXTENDING THE COMDEF UML PROFILE ... 54
6.7.1 Application ... *55*
6.7.2 Contains ... *55*
6.7.3 Dialog ... *55*
6.7.4 Extending CML ... *55*
6.7.5 Application CML grammar extension ... *55*
6.8 MODEL LAYOUT CLASS- GENERATOR (EMITTER) ... 56
6.9 SUMMARY ... 57

CHAPTER 7 MULTIDEV MODELS. ... 59

7.1 DOMAIN MODEL ... 59
7.2 TASK MODEL ... 60

7.3 THE PRESENTATION MODEL 62
7.3.1 Dialog meta-model 63
7.3.2 Dialog metamodel main concepts 65
7.3.3 Dialog object properties 71
7.3.4 Mapping the tasks to presentation 71
7.4 RELATIONSHIPS BETWEEN MODELS 71
7.5 SUMMARY 72

CHAPTER 8 DIALOG MODELING AND CODE GENERATION 73

8.1 MULTIDEV MODELING TOOL 73
8.1.1 Modeling environment 73
8.1.2 Modeling concept 74
8.2 CODE GENERATION 74
8.2.1 UIML mapping 75
8.2.2 Layout algorithm 75
8.3 CLIENT APPLICATION LOGIC 76
8.3.1 Connection to the user interface 78
8.4 SUMMARY 79

CHAPTER 9 CASE REVISITED 81

9.1 THE CAR RENTAL EXAMPLE 81
9.2 CODE GENERATION 85
9.3 RELATIONSHIP TO THE DISTRIBUTED SYSTEM 87
9.4 SUMMARY 88

CHAPTER 10 CONCLUSIONS, FURTHER WORK AND SHORTCOMINGS . 89

10.1 EVALUATION OF MULTIDEV AGAINST REQUIREMENTS OF CHAPTER 3 89
10.1.1 Productivity 89
10.1.2 Model concept requirements 90
10.1.3 Multi platform support 91
10.1.4 Evolution 91
10.2 CONCLUSIONS 91
10.3 FURTHER WORK AND SHORTCOMINGS 92
10.3.1 Inherited COMDEF shortcomings 92
10.3.2 Task model 93
10.3.3 Wider platform support 93
10.3.4 Extending the COMDEF UML profile 93
10.3.5 Multidev Modeling tool 93
10.3.6 Possible Multidev evolvement - COMDEF 93
10.3.7 Possible Multidev evolvement - UIML 93
10.3.8 Latest Model Based Developments and Multidev 94
10.3.9 OMG and MDA 94
10.3.10 Software Factories and Microsoft Domain Specific Language 96
10.3.11 Multidev and MBD's new approaches 97
10.4 SUMMARY 99

APPENDIX A THE CAR RENTAL SYSTEM AND COMDEF 101

APPENDIX B AN UIML EXAMPLE FILE 105

APPENDIX C THE COMPLETE CML GRAMMAR 109

APPENDIX D BIBLIOGRAPHY 113

APPENDIX E INDEX 119

List of Figures

Figure 1.1 - The COMDEF development process 16
Figure 1.2 - COMDEF conceptual architecture........ 16
Figure 2.1 – The car rental system model 19
Figure 2.2 – The use cases of the Car Rental System........ 22
Figure 4.1 – Genova object selection user interface 30
Figure 4.2 – Genova object selection user interface 31
Figure 4.3 – The dialog designer tool........ 32
Figure 4.4 – Mobi-d development cycle........ 35
Figure 5.1 – The COMDEF conceptual architecture........ 42
Figure 6.1 – Multidev development process overview........ 48
Figure 6.2 – Multidev conceptual architecture 50
Figure 6.3 – Deployment of user interface through a renderer........ 51
Figure 6.4 – deployment of user interface trough an interface server 51
Figure 6.5 – Conceptual metamodel........ 54
Figure 6.6 – The emitter process 57
Figure 7.1 – Screen shot of the domain model from Multidev modelling tool. 60
Figure 7.1 – The task model for retrieving a stock value. 62
Figure 7.2 – The Multidev Dialog Metamodel 64
Figure 7.3 – The dialog object and its associations. 65
Figure 7.4 – The block object. 65
Figure 7.5 – General block. 66
Figure 7.6 – Window block components. 67
Figure 7.8 – Data Item. 68
Figure 7.9 – Button block. 68
Figure 7.10 – List block. 69
Figure 7.11 – The tabbed pane component. 69
Figure 7.12 – The tree view object. 70
Figure 7.13 – The menu components 70
Figure 7.14 – The toolbar. 71
Figure 8.1 – A screen shot of the user interface tool 74
Figure 8.2 – A login dialog........ 75
Figure 8.3 – The presentation model for the user login dialog........ 76
Figure 8.4 – A presentation model for the task model 77
Figure 8.5 – Class overview 77
Figure 8.6 – The Domain, task and presentation model for stock value 78
Figure 8.7 – D-T-P models with render........ 78
Figure 9.2 - The Dialog model 82
Figure 9.3 –the Domain model 83
Figure 9.4 – The dialog model 83
Figure 9.5 – the dialog elements toolbar. 84
Figure 9.6 – The data item definition dialog. 84
Figure 9.7 – Task model 85
Figure 9.8 – The initial menu of the client 85
Figure 9.9 – UIML definition of a button........ 86
Figure 9.10 – UIML example of behavior definition. 86
Figure 9.11 - the new customer interface example........ 86
Figure 9.12 – Java code calling the renderer. 87
Figure 10.1 - MDA Multidev process similarities. 98
Figure A1 – The COMDEF development process 101
Figure A2 – A CML Entity example 102
Figure A3 – Generated Java code for Reservation Entity 103

Chapter 1

Introduction

This chapter introduces the problem area and the motivation of the thesis. Background information and the context in which this thesis is written are presented. At the end of the chapter an overview of how the thesis structure is offered.

1.1 Background

Building user interfaces is time consuming and hence costly. In a paper presented by Myers and Rosson [Myers, 1992] a survey concludes that nearly 50 % of source code lines and development time is related to user interface building. This suggests that there should be possible to reduce the over all development time by finding ways to automate the process of generating user interfaces.

Today almost every computer is hooked up to some kind of network. At the time of writing this thesis an estimated 1.1 billion users have access to the Internet [World stats, 2006]. The impact of network technology has changed the computer industry from focus on single user applications running on one machine to multi user and multi platform systems. Because of this change, new technologies have been developed to utilize the distributed environment that the waste usage of networks offers. This introduces new challenges for the developers of distributed systems. New frameworks and methodologies have submerged to meet these challenges. However, even if the setting has changed, some properties are still as important as ever, such as low over all cost of software development. As stated, the user interface is a big part of the over all system development process. This leads to the goal of this thesis which is to present a framework that helps reduce the time of developing clients to distributed information systems, which includes the client application and the user interfaces, and at the same time pay respect to the properties all ready set by the developing environment for a distributed computer system.

1.1.1 Research in graphical user interfaces

There has been extensive research in the field of graphical user interfaces. This research has been done from different viewpoints, which can be divided into four different areas [Machiraju, 1996]. The first deals with the study of interfaces themselves, their properties and the design options that can be adopted when developing user interfaces. The second area exploits the different methods for designing tools to develop efficient user interfaces. The third area looks into the different interactions techniques in user interfaces, such as input devices and other hardware devices. The last identified area in user interface research tries to analyze existing user interfaces and tools that can aid this analysis. All of these areas do intervene but this thesis will reside in the second identified area that investigates the development of tools to develop user interfaces.

There are many tools developed that aid in the task of developing user interfaces, many with different viewpoints. These tools range from semi-automatic to automatic generation of user interfaces. They also differ in which user group they target in regards to how skilled the user needs to be in software development. The following sub chapters will present different categories of tools and technologies in this field of research divided into low level- and high-level tools. The separation between low- and high-level is done by the abstraction level they offer. Lower abstraction level often yields a higher degree of programming effort

by the developer. The purpose of the presentation is to place the framework presented in this thesis on the map of user interface research.

1.1.1.1 Low level tools

Windowing systems such as Microsoft Windows and Macintosh can be viewed as user interface tools because they provide procedures (API) that applications can use to draw themselves on the screen. On top of the windowing systems, we find toolkits. A toolkit is a library of “widgets” that can be called by the application program. A widget is a user interface component such as menus, buttons, scrollbars, text fields etc. Toolkits provide the advantages of the component methodology, which has been used successfully by other industries for years. The advantage of this methodology is that once a component is developed it can be reused. This reduces cost and in the case of user interfaces, it ensures that different applications that use the same toolkit will act and look similar. Examples of toolkits are Java AWT [Microsystems, 2001 #1], java SWING libraries [Microsystems, 2001 #2] from SUN Microsystems. Using toolkits is not trivial and requires programming skill and knowledge in programming languages. Toolkits are low-level tools, because they require a considerable programming effort.

1.1.1.2 High level tools

Programming at toolkit level is difficult. Therefore many higher-level tools have been developed that helps the developer ease the production of user interfaces. Much research has been done on high-level tools, and there are many different strategies. The following sections will introduce the different categories of these tools.

1.1.1.2.1 Language based tools

Language based tools describe user interfaces in different languages, such as event-, declarative- and constraint languages. The languages are used to describe the syntax of the user interface. The description is then compiled to produce the actual user interface. The language-based tools require that the user interface designer have programming experience. Some programmers have objected to learn a new language just for programming user interfaces [Olsen, 1987].

1.1.1.2.2 Application frameworks

Application-frameworks are another branch of high-level user interface tools. These frameworks are developed to help the programmer to use the different toolkits that the framework supports. They do this by providing classes for the important part of an application such as the main window, the commands etc., and the programmer can then specialize these classes for the different applications. Frameworks exist for all major platforms such as Microsoft foundation classes for Windows (MFC) [Microsoft, 2001], and the CodeWarrior PowerPlant for Macintosh [Metrowerks, 2001]. Although these frameworks bring the usage of toolkits up to a higher level of abstraction, they still require that the designer of the user interface has solid programming knowledge.

1.1.1.2.3 Interface builders

Other high level tools provide interface builders, often as a part of a programming environment. Tools such as Visual café from Symantec [Symantec, webgain 2006] and Visual studio from Microsoft [Microsoft, 2003 #1] provide interface builders that allow designers to select components from libraries. The interface is visualized in the interface builder. The interface builder will generate skeleton code, which has to be implemented. In the same category are tools such as Microsoft Access [Microsoft, 2003 #2] that allows the designer to design user interfaces from the included components, without implementing any code.

1.1.1.2.4 Model-based automatic generation

Another approach in high-level tools is model-based automatic generation. These tools automatically generate the user interface from a higher-level specification of the user interface layout. The different tools vary in what way the model is generated. Examples of such tools are UIDE [Sukaviriya et al., 1993], Humanoid [Szekely et al., 1993] and MASTERMIND [Neches et al., 1993]. Tools that generate user interfaces have been criticized [Szekely, 1996] [Puerta, 1996] for generating too simple user interfaces, and that the specification languages are hard to learn and use.

1.1.2 Model- and component based software development

Software engineering became a term in the late 60's. Since then, there have been many efforts to increase quality and productivity of software development. In the last years, the trend has focused on using engineering techniques that has been used by traditional engineering disciplines for years. These are model- and component-based development. Model-based software development has proven itself to be useful because it helps the developer or developer team to better understand complex systems. By breaking the system down into different models, it becomes more comprehensible and thereby many pitfalls can be avoided. If a methodology that supports modeling is used, the process of developing software is less likely to take an *ad hoc* approach. This can ensure that the models are sound and that all partial models of the system together describe the whole system in detail. UML [UML, 2005 #1] [UML, 2005 #2] is a modeling language intended for object-oriented modeling. Object-oriented modeling means that the main building block of every part of the system is an object (or a class). The purpose of UML is to visualize, specify, construct and document object oriented systems and has become the industry standard tool for object-oriented modeling.
Component software development focuses on building software systems from finished software components. The advantages of component based software development are many. By reusing components, the overall cost of the finished product decreases and the quality (may) increase. In software development, a component is a piece of software that delivers a service and has a well-defined interface. A software system can then be built by combining these components.

1.2 COMDEF

COMDEF [Kvalheim, 1999] is a framework for component development, which is intended for developing components in a distributed environment. This framework is described later in this thesis, but an introduction is given here in order to understand the relationship between the work presented in this thesis and COMDEF.

COMDEF provides a model-based framework for developing distributed component based systems, which shifts the development focus from the implementation to the modeling of a system. This is done through automatic code generation from a graphical (UML) – or a lexical (CML - Component Modeling Language) model of the system. The original COMDEF development process is based on using a methodology that produces a UML model of the system. This is done by modeling with the COMDEF metamodel, which provides stereotypes for the different modeling parts of COMDEF. The model is then converted to a lexical language (CML). This representation is the bases for code generation of the system. By applying various code-generation tools (emitters) to the CML code, most parts of the system are generated. There are different emitters for different distributed architectures such as EJB, CORBA or COM. Figure 1.1 shows an overview of the COMDEF development process.

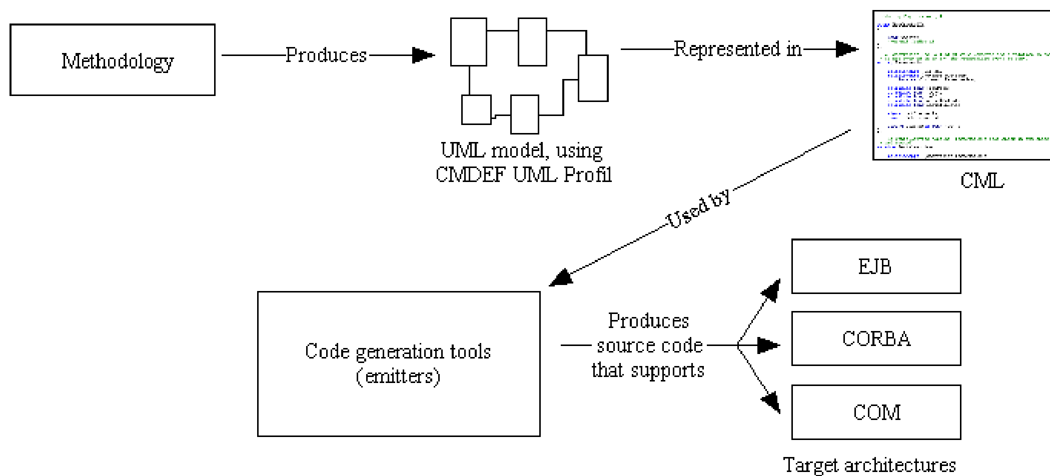

Figure 1.1 - The COMDEF development process

The conceptual architecture of COMDEF is depicted in figure 1.2. It shows the main building blocks of the framework. The architecture can be divided into three different main layers:

- **Application layer** – the client and the userservice
- **Business layer** – the service and the entities
- **Data layer -** the persistent storage

The system is modeled in UML using the COMDEF UML profile, which is an extension to the UML metamodel and provides stereotypes for the different building blocks shown in figure 1.2.

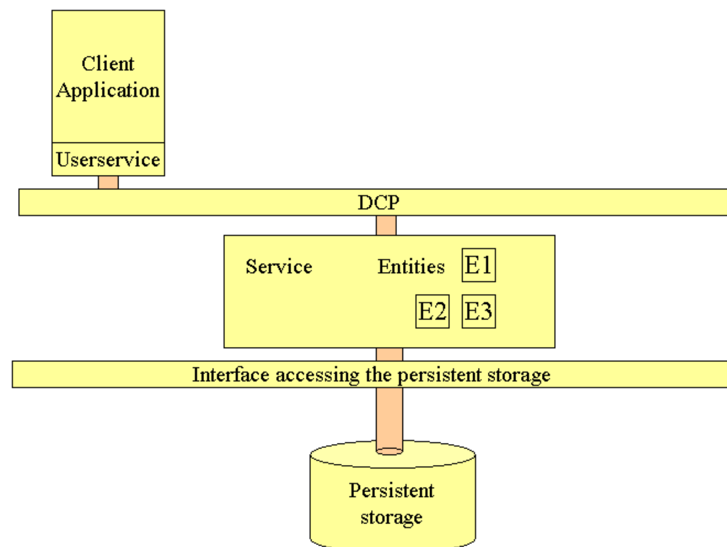

Figure 1.2 - COMDEF conceptual architecture

As figure 1.2 shows, the application layer consists of the user service and the client application. The userservice is the client's interface to the services. There can be many userservices and a userservice can be the interface for many services. In COMDEF, the client part of the system is described as a client pack that consists of a client program and userservice package/packages. The userservice is an abstraction of the DCP so that it is transparent to the client what kind of underlying architecture that is used.

1.3 Context of thesis

This thesis presents a framework for generating clients to computer systems that are based on COMDEF. The framework is called Multidev and is a model-based framework for developing multi platform clients in a distributed environment. The distributed environment in this context is systems that are developed with COMDEF. Multidev tries to utilize the information in the UML model of the system to generate as much of the client as possible, with as little intervention by the developer as possible. The Multidev framework includes a high level tool with a graphical environment for user interface modeling that lifts the abstraction level from implementation to modeling of the user interfaces. This tool fits under the label of *Model-based automatic generation* described in 1.1.1.2.4.

Multidev presents a model-based strategy on several levels. The framework is an extension to the COMDEF framework, and bases the user interface generation on the COMDEF UML profile. The tool provided in the framework also lets the user handle the specification of the user interface layout on a higher level of abstraction then language-based tools. This is achieved by providing a graphical model of the user interface layout. A key point is also that by describing the layout of the user interface once, it should be possible to map the client application to many different platforms of different thickness. Thickness in this context is described as the functionality of the client platform. Typical a java application is a thick client; a web application is a medium thick client, while a thin client is typically a cellular phone with a WAP client.

COMDEF was an early attempt at Modeling Based Development. While working on this thesis, the development of COMDEF has stopped. However, there are many new initiatives that provide similar frameworks, such as MDA from OMG and DSL from Microsoft. (Both of these are described in chapter 10)

Multidev was developed as an add-on to the COMDEF framework, but the focus of the Multidev framework is independent of the actual framework used. Therefore, with the experience from COMDEF as a base, it should be possible to transfer the Multidev framework to similar frameworks, such as the OMG MDA and/or DSL from Microsoft.

1.4 Structure of thesis

The structure of the thesis is presented in the list below.

Chapter 2 Case study -This chapter introduces a case study of a system developed using COMDEF and introduces the task that Multidev seeks to solve.

Chapter 3 Requirements - This chapter defines the requirements that is to be fulfilled by the framework presented in this thesis

Chapter 4 Evaluation of existing work - This chapter looks on existing work. A brief description is presented and how they can solve the problems presented in the case study. Shortcomings and problems are defined.

Chapter 5 Multidev's technology relationships - This chapter presents different software technology that either is used in the development of Multidev or is used by Multidev to produce the COMDEF clients.

Chapter 6 Multidev - This chapter gives an over view of Multidev and presents its relationship with the COMDEF framework.

Chapter 8 Multidev models - This chapter presents the three different models that Multidev uses to construct the COMDEF client

Chapter 8 Dialog modeling and code generation - This chapter elaborates on how modeling is done in the Multidev framework and how the code generation is accomplished.

Chapter 9 Case revisited – The case is revisited and Multidev is applied to the case.

Chapter 10 Conclusions, further work – The framework of this thesis is evaluated with respect to the requirements from chapter 3. Further work is identified and solutions to problems are suggested. In addition the two main (at the time of writing) paths of Model Based Frameworks are presented and discussed in view of Multidev.

1.5 Summary

This chapter has given background information as well as motivation for this thesis. An introduction to user interface research has been presented to place Multidev on the map in this field of research. The COMDEF framework for building component based distributed computer systems was presented to show the relationship between Multidev and COMDEF. An overview of the structure of the thesis was also presented.

Chapter 2

Case study

In order to understand the problem for which this thesis suggests a solution, a test case was developed. The case is a system for renting cars to customers, called the Car Rental System. The model of the system is shown in figure 2.1 and is a UML model based on the COMDEF framework. This framework will be described in more detail in chapter 5.

The Car Rental System is not meant to be a fully functional system ready to be used in a business, but is used to illustrate what is needed of a client that run on a COMDEF based system. The case study presents the case and describes some general criteria to the solution. Appendix A describes the implementation of the system in the context of COMDEF.

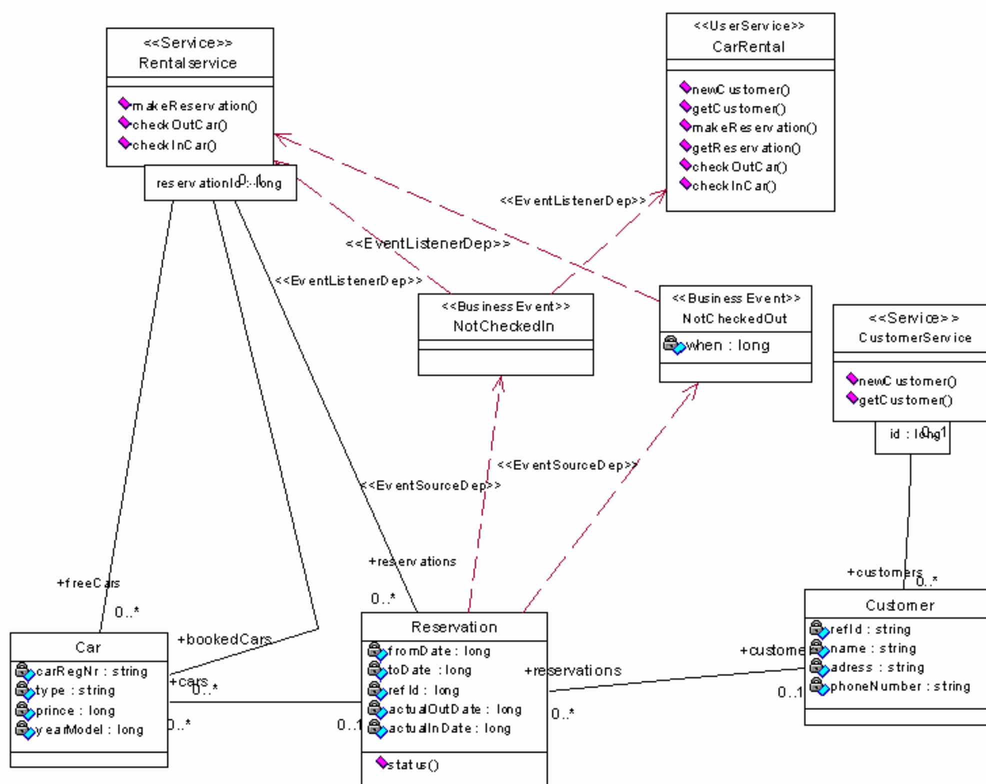

Figure 2.1 – The car rental system model

The Car Rental System registers and maintains data about cars, customers and reservations of cars. As mentioned, it is based on the COMDEF framework. This is a framework for developing component based distributed systems. In COMDEF, a key point is to code-generate as much of the systems source code as possible. However, the COMDEF framework does not generate any source code for the application tier of the system, except for the userservice.

2.1 The Car Rental model elements

As the figure 2.1 shows, the system is built up by entities, services, events and a userservice. Entities are an abstraction of data elements and are persistent objects that logically are contained inside services. The different entities are described bellow to help get a overview of what the user interface of the client should be able to display to the user and pass down to the user service from user input. The services and the user service are also described in the following sections.

2.1.1 Car entity

Instances of the car entity represent the different cars managed by the system. The car entity stores information about cars in the following attributes

Attribute name	Type	Explanation
CarRegNr	String	The registration number
Type	String	The type of car
Price	Long	The rental price
YearModel	Long	The year model of the car

The car entity is logically a part of the rental service, which keeps track of the free and booked cars. As figure 2.1 shows, the reservation entity also has an association to the car entity. This association means that a car is reserved by a reservation

2.1.2 Reservation Entity

The reservation entity stores information needed for a reservation in the following attributes:

Attribute name	Type	Explanation
FromDate	Long	The date the reservation was made
ToDate	Long	The date the reservation expires
Refid	Long	An identifier for this reservation
ActualOutDate	Long	The date the car was actually rented out
ActualInDate	Long	The date the car was actually returned

In addition, the reservation entity has one operation:

Operation name	Return type	Parameters	Explanation
Status	String	None	The status of the reservation

Instances of this class represent the reservations made in the Car Rental System. The reservation class has associations to both the car and the customer classes. A reservation is made by a customer and is a reservation for one or more cars.

2.1.3 Customer entity

The customer entity holds information about a customer in the following attributes:

Attribute name	Type	Explanation
RefId	String	An identifier for the customer
Name	String	The full name of the customer
Address	String	The customers address
PhoneNumber	String	The customers phone number

The customer class has an association to the reservation entity. A customer may have one or more reservation at a given time. Logically the customer entity resides in the customer service.

2.1.4 Services

There are two services in the model described in figure 2.1. A service is a component from a COMDEF viewpoint. Services have a specified task, or service that they offers to the clients through the user service. In the Car Rental System, there are two services, the customer service and the rental service:

Customer service:
The customer service is responsible for customer handling, such as registering new customers and find existing customers. The Customer service has three operations as figure 2.1 depicts.

Rental service:
The rental service is responsible for reservation of cars. This includes making a new reservation and book a car for a given customer for a given time period. The customer service has two operations as figure 2.1 depicts.

2.1.5 User service

The user service is the glue between the services and the client application. It provides a transparent view to the services for the client application. Transparent in this context means that the client application should not be aware of what kind of distributed computing platform that is used. The user service in the Car Rental System provides the operations of both services to the client application.

2.1.6 Events

The car rental system has two events. These *are not checked* in and *not checked out*, as figure 2.1 depicts. Both of these events are triggered from the reservation service if the car is not checked in, or not checked out according to the given dates.

2.2 Use cases

The Car Rental System is developed using a use case driven methodology. This means that use cases are identified in order to produce the requirements that that the system must

satisfy. The different use cases are depicted in figure 2.2 and described in the following sections.

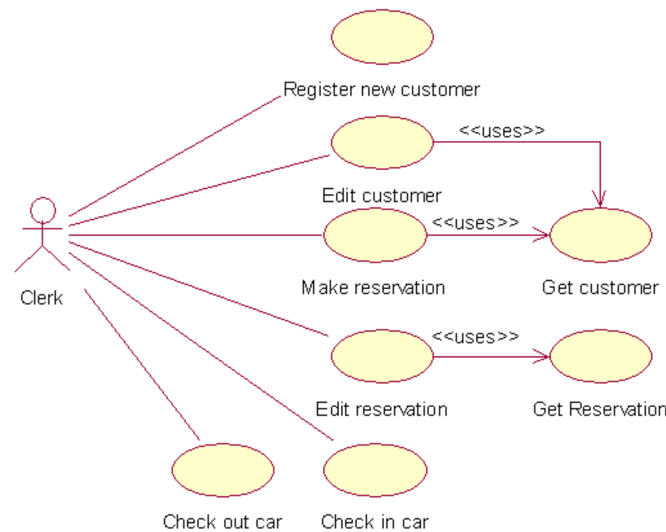

Figure 2.2 – The use cases of the Car Rental System.

2.2.1 Register new customer

This use case describes the task of adding a new customer to the system. It has the following description.

Primary actor: The clerk.
Goal: Register a new customer in the system.

1. Get customer information.
2. Submit information to system.

2.2.2 Get customer

This use case describes the task of finding a customer in the system. It is used by other use cases to complete their goal. The use case has the following description:

Primary actor: The clerk
Goal: Find an existing customer.

1. Submit Customer identity
2. Get customer object

Extensions:
1. No customer with the submitted identity

2.2.3 Edit customer

This use case is used to alter customer information, such as address and telephone number. It uses the *get customer* use case.

Primary actor: The clerk.
Goal: Edit customer information

1. Execute the get customer use case to get a customer.
2. Edit the customer information
3. Submit information to system.

2.2.4 Make reservation
This use case is used to make a new reservation of one or more cars. It uses the *get customer* use case to get a customer. The make reservation use case has the following description.

Primary actor: The clerk
Goal: Make a new reservation of one ore more cars for a customer
1. Execute the get customer use case.
2. Submit data information
3. Associate one or more cars with the reservation
4. Submit information to system.

Extensions:
1. The customer is not in the system.
   1a Execute the new customer use case.

2.2.5 Get reservation
The get reservation is used to find an existing reservation. It is used by the *edit reservation* use case. The get reservation use case has the following description:

Primary actor: The clerk
Goal: Get an existing reservation from the system.
1. Submit reservation ID
2. Get reservation object

Extension:
1. No such reservation in system

2.2.6 Edit reservation
The edit reservation use case is used to edit an existing reservation. This use case uses the *get reservation* use case. The use case has the following description:

Primary actor: The clerk.
Goal: Edit an existing reservation.
1. Execute get reservation use case
2. Edit reservation information
3. Submit information to system

2.2.7 Check out car
This use case is used to check out the car when the customer actually checks out the car. The use case uses the *get reservation* use case. The check out car use case has the following description.

Primary actor: The clerk
Goal: Check out one ore more cars in a reservation
1. Execute get reservation use case
2. Check out the car(s)

2.2.8 Check in car
This use case is used to check in the car when the customer delivers the car back. The use case uses the *get reservation* use case.

Primary goal: The clerk
Goal: Check in the cars of a reservation
1. Execute the get reservation use case

2. Check in the car(s)

2.3 General client requirements

The client is the application layer described in section 1.2. The main responsibilities of the application layer are to connect to the services through the user service and to the end user through the user interfaces. The user service provides the operations of the services to the client application and the logic of how these operations are used must be handled by the client.
If the distributed nature of COMDEF is to be upheld by the framework presented in this thesis, the framework must have the ability to generate clients for multiple platforms. A platform in this context is defined as the combination of hardware device and operating system. The different clients must provide the same functionality on all supported platforms.

2.4 Problem areas identified

The only alternative that exists to day for developing the client application with the existing version of COMDEF is to develop it by hand. This section presents the problems and shortcomings of solving the case by hand implementing the client.

Productivity
The system should be able to work on multiple platforms. This would mean that the client for each platform has to be coded by hand and requires that the developer have to be familiar with many programming languages and platforms.

System evolution
The client is the highest tier in a multi tiered architecture. If the model of the underlying architecture changes, then there is a big chance that the client will not work. The changes to the client have to be coded by hand. If the system is to be supported on many platforms, then even small changes to the model could result in a huge task to update the clients to work with the changes.

2.5 Summary

This chapter has described a typical case that the framework of this thesis seeks to solve. The case is a car rental system developed using the COMDEF framework. The task is to build a client that can be used with this system, with as little effort as possible by the developer. This chapter has presented the details of the system and what is needed of a client that will run on it. The Car Rental System consists of three entities, two services and a userservice. The client application needs to address both the communication with the user service as well as communication with the user through graphic user interfaces. The chapter also defines some general requirements to the client application and defines the problem areas that must be solved by the framework described in this thesis.

Chapter 3

Requirements

This chapter identifies the requirements to Multidev. These requirements are based on the problems identified in the background chapter and case study chapter. Other requirements are also added.

The requirements are dived into the following areas

- **Productivity**
- **Modeling concepts**
- **Multi platform support**
- **Evolution**

Each of these areas has requirements that will be discussed in the following sections.

3.1 Productivity

The task of the framework described in this thesis is to aid the development of clients to systems build with the COMDEF framework. The clients must support the architecture defined in COMDEF for distributed information systems (DIS). Besides that, the framework must produce clients that can run on different platforms. If this is to be done by hand coding the different clients, it will be a vast job. Each client will be a separate project. The framework should provide tools that simplify the process and increase the productivity of the development. In table 3.1 the productivity requirements are listed and in the following sections the requirements are discussed:

Table 3.1 Productivity requirements

Category	**Requirement**
Code generation	**P1** - Code generate the generic code **P2** - Code generate user interface **P3** - Code generate from graphical notation
Independency	**P4** - Model environment independent
General	**P5** - Graphic editable model of GUI **P6** - Template support **P7 -** Developer control **P8** - Support for third party gui components **P9** - Support interface guidelines

3.1.1 Code generation

Hand programming gives the advantage of flexibility. However, it is a time consuming and often error prone job. In order to let the developer focus on functionality instead of programming techniques, the development should be model based. All generic code should be generated, so that the developers don't need to focus on technical issues, such as connection to the DIS (P1).

Besides generating the generic code, the, framework should generate the user interfaces as well. The framework should provide a model of the user interface from witch it can be

generated (P2). To avoid error prone lexical models, the user interface should be generated from a graphical notation.

3.1.2 Independency

There are many modeling tools on the marked today. UML modeling tools range from free software with little functionality, to expensive software with options beyond comprehension. Apart from supporting the UML modeling language many of the different products are proprietary and with little support for sharing the models between them or to export or import the UML models to or from other tools.

The OMG (Object Management Group) [OMG, 2006] has developed a XML based language for metadata interchange that can be used to export and import UML models as well. This format has been named XMI [Frankel, 2001] and stands for XML Metadata Interchange. The XMI language will be elaborated in chapter 5.

Every tool that supports XMI has the advantage that they can export and import data from other tools that support the same branch of XMI. In order to be as independent as possible in regard to the environment where the COMDEF system model is developed, the framework should import the UML model as XMI, defined by a specific DTD (P4).

3.1.3 General

This section describes other requirements to increase the productivity of the client development.

As stated in chapter one, it is difficult to generate good user interfaces in a fully automated process, so the framework should allow for developer control (P7).

The developer can intervene many different ways. Graphical editable models would let the developer concentrate on the modeling (P5), instead of low level programming. There are other strategies available to let the user intervene in the development process. One such strategy is to have lexical description of the models of the user interface and application logic, but since this requires the developer to be familiar with, or have to learn a new language, the use of graphical models is preferred.

There are many different ways to display a user interface. Different developers like different layouts. Besides generating the user interface, the framework also should suggest layout patterns that the developer can use as a basis. By providing different patterns, a developer can choose a pattern that conforms to his or her likings, or the best fit for the task that the user interface shall accomplish (P6). These patterns will also contribute to upholding different interface guidelines (P9).

Most user interfaces are based on well-known user interface parts such as text boxes, labels and so on. However, not all user interface needs can be covered by simple interface objects. Some objects may be made especially for a task of a particular client, but can be reused in other clients. The framework must provide a way for adding new user interface components to its repository of user interface objects (P8).

3.2 Modeling concepts

Today many different technologies are supported on many different platforms. The different platforms can present their user interface in different ways. As an example, consider Lynx, a text based web browser. Lynx will present everything in a web page as text. Pictures are presented as links to an external picture viewer. List boxes are displayed as a set of

hyperlinks. Still, a web page presented in Lynx contains the same information that the page would if it was presented in graphical browser such as Netscape.

To be able to model the user interface without knowing the clients platform, the framework should provide a metamodel of the user interface objects with predefined mappings to the supported platforms. The mapping against the specific platform should happen as late as possible in order for the model to be generic until code generation. The user interface model will be a separate model, although it will be closely linked to the model of the underlying system.

Another aspect of the client, that the framework must address, is the information flow between the client application, the user interfaces and the different tasks the user interfaces are defined to address.

Table 3.3 shows the different model concept requirements, and the following sections describe them.

Table 3-2 – Model concepts requirement

Category	Requirement
General	**M1** - Model based framework **M2** - Understand and use the COMDEF metamodel/ UML profile **M3** – Declarative models
Architecture	**M4** – User interface and application logic in separate tiers

3.2.1 General

The main reason for wanting a model-based framework is to lift the abstraction level of the developing process from implementation to modeling (M1).

The framework will read its domain data from a UML model. This model is always based on the COMDEF metamodel, and so the framework must understand the different concepts this metamodel describes (M2). A part from getting domain information from the COMDEF UML model, the framework should also provide a tool for modeling the structure and functionality of the client using declarative models (M3).

3.2.2 Architecture

As mentioned, the task of the framework presented in this thesis is to develop clients to COMDEF systems. This framework has a well-defined architecture. The only client specific part of this architecture is the user service. The framework should provide a clear architecture for the client that build on the COMDEF framework.

The clients should also be able to run on different platforms, and these platforms may have different ways of displaying user interfaces. Because of this, the framework should provide a multi tier client architecture that separates the user interface and the client application logic (M4).

3.3 Multiple platform support

As stated, there are many technologies used on many different platforms. In order to fully use a distributed information system, the system should be accessible from more then just

desktop computers. Mobile networks, such as UMTS [Wikipedia UMTS, 2006] will give the mobile range of appliance good enough band with to be highly functional client platforms to a distributed information systems, making them target platform for COMDEF clients. The COMDEF framework is built to support a distributed environment, and therefore have no limitation on what kind of platform the system client is running. The clients can vary in thickness from thin (wap, web-applications) to medium sized (java-apples) to thick clients (java, c++).

Table 3-3 - Multiple platform support

Category	Requirements
General	**MPS1** – Multi platform support

3.4 Evolution

Almost every system developed will require updates and changes as time goes by. Technology changes and the requirements of the user changes. A small change in the DIS could mean that every client has to be redeveloped. Since changes inevitably will occur it is important that these can be reflected and incorporated with as little effort as possible.

Table 3-5 - Evolution

Category	Requirement
Evolution	**E1** – Support changes in an easy manner

3.5 Evaluation criteria

The requirements defined in this chapter will be used to compare different solutions. To be able to do this comparison, the tables of requirements will be presented in appropriate places trough out the thesis with values that will rate their solution against the requirements. The values use will be:

- 2 supported.
- 1 partially supported.
- 0 not supported.

3.6 Summary

In this chapter, the different areas of requirements have been structured in to measurable requirements. These areas are productivity, modeling concepts, multi platform support and evolution. The different requirements are structured into four different matrixes that will be presented trough out the thesis where appropriate.

Chapter 4

Evaluation of existing work

This chapter will look at existing technologies and see how well they perform on the case described in chapter two, with respect to the requirements defined in chapter three.

The technologies that will be evaluated are:
- Genova
- Mobi-d
- Visual Café

The selected tools serve as representatives for different groups of technology in user interface research, as presented in chapter 1.

4.1 Genera Genova overview

Genova is a commercial product from Genera [Genera, 2000 #1]. Genova is a development environment that focuses on development and maintenance of information management systems. The development process that is proposed when using Genova should be architecture-centric, use case driven, iterative and incremental [Genera, 2000 #2]. Genova does not include an object-oriented modeling tool, but it is integrated with a selected range of modeling tools such as Rational Rose [Wikipedia Rational, 2006] from Rational Software Inc and Select Enterprise [Select, 2006] from Select Software tools. Genova has two major areas of functionality. It can be used to make the object model persistent. This is done with the DB Designer tool, which can transform the object model into two-dimensional relational tables. DB designer also lets you generate compile and run the application environment that glues together the application with the database. The other function of Genova is to model and generate user interfaces based on an object model. When using Genova with a modeling tool it supports, such as Rational Rose, it imports the model and provides an environment where the designer can manipulate and generate the user interface for a selected range of platforms.
According to Genera, the concept behind Genova is not to develop a finished product, but is to be used for rapid prototyping to reveal missing model information [Vogt, 2001].

At the start of working with this thesis the Genova was in version 6.1. The latest version of the software was released in December 2004 and is version 8.1. Genera as a company has bin merged with Software Innovation ASA. The product development part has been established as a new company called Software Innovation Esita [Software Innovation]

4.1.1 Genova work protocol

The starting point with the methodology proposed with Genova is to make use cases that describe the total functionality of the system. These use cases are then refined into textual descriptions (goal, pre-condition, sequence of activities, post-conditions). Based on the use case model, an iterative process of making an object model is started. For every iteration, the Genova tool is used to make prototype of the user interface. According to Genera, this will help making the object model more complete because missing information in the object model is easier detected.

The prototype is developed with an activity called dialog modeling. The dialog model is constructed by making an object selection from the UML object model. Objects that

together fulfill a use case are selected and saved in the Genova repository, for later use in the dialog designer tool. The selection is visualized as a tree view. Classes that are linked together by relationships can be included and this information is used by the dialog designer to make a proposal of what kind of user interface objects that will represent the classes. Conceptually the selection of objects does not contain classes, but the role the classes play in the use case. The figures below show the user interfaces used when making an object selection.

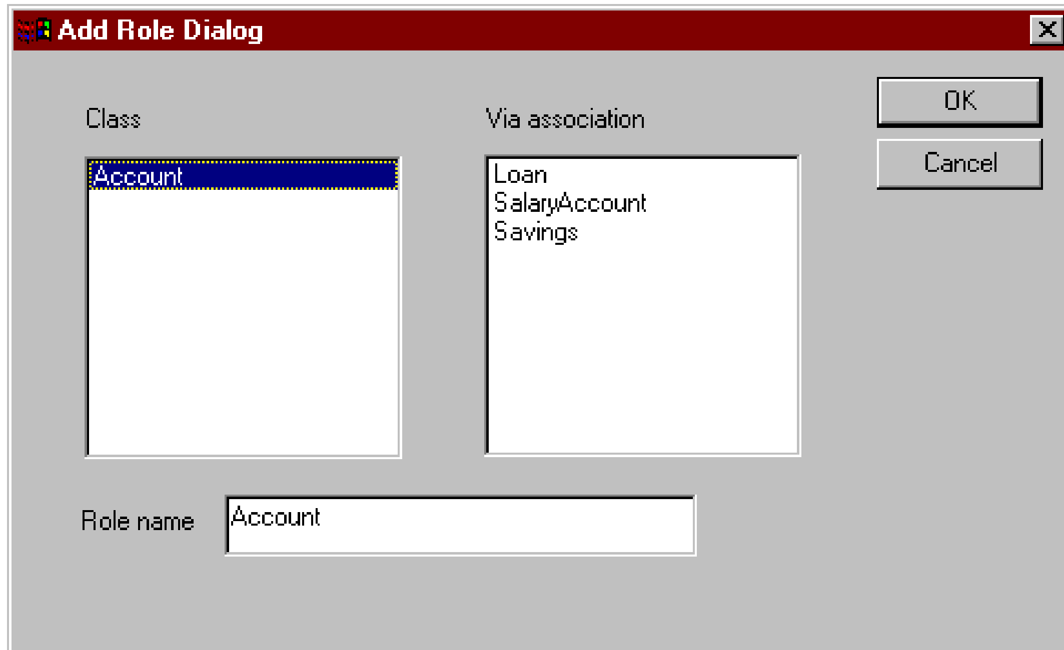

Figure 4.1 – Genova object selection user interface

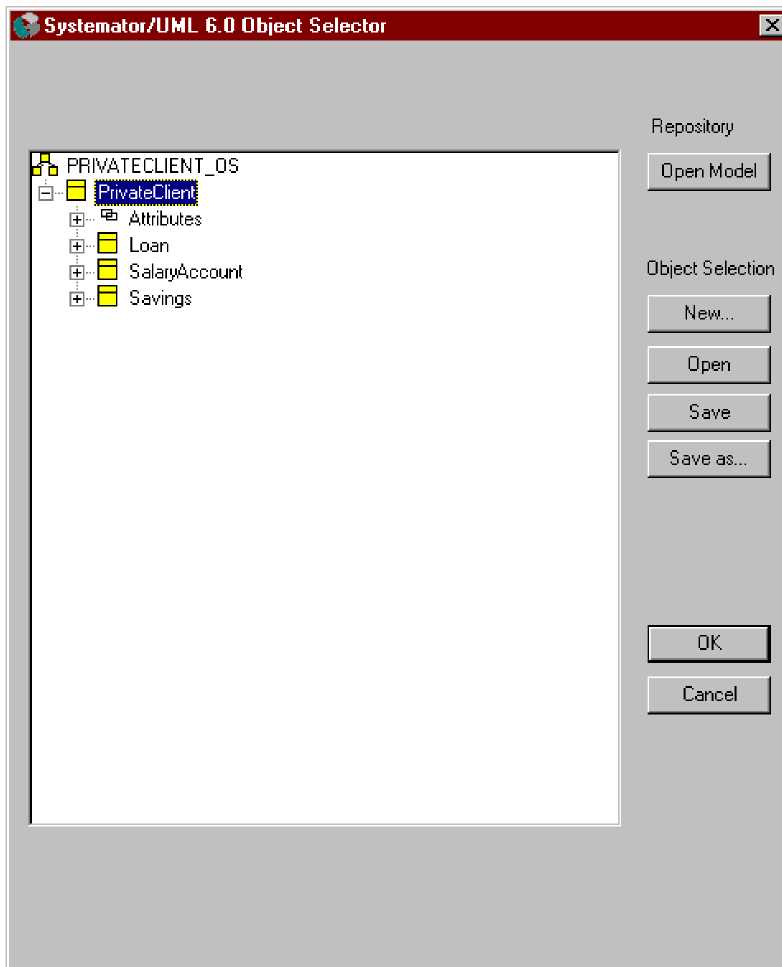

Figure 4.2 – Genova object selection user interface

When the object selection is complete, the next step is to generate the user interface. A style guide is used to define how the different model elements are mapped into user interface constructions. The tool will suggest a layout depending on the style chosen and the objects selected. As an example, a class that as a one-to-many relationship with another class can be represented with a set of data fields for the root and a scrollable list box for the related class. This makes it possible to scroll through the related object instances of the root object.

The structure of the user interface is shown in a topological manner as a tree view in the dialog designer tool. This tool lets the developer work on the dialog model using an object palette to add new user interface objects, and let him or her set properties for the different objects in the use interface. The layout of the dialog model is shown in the figure bellow.

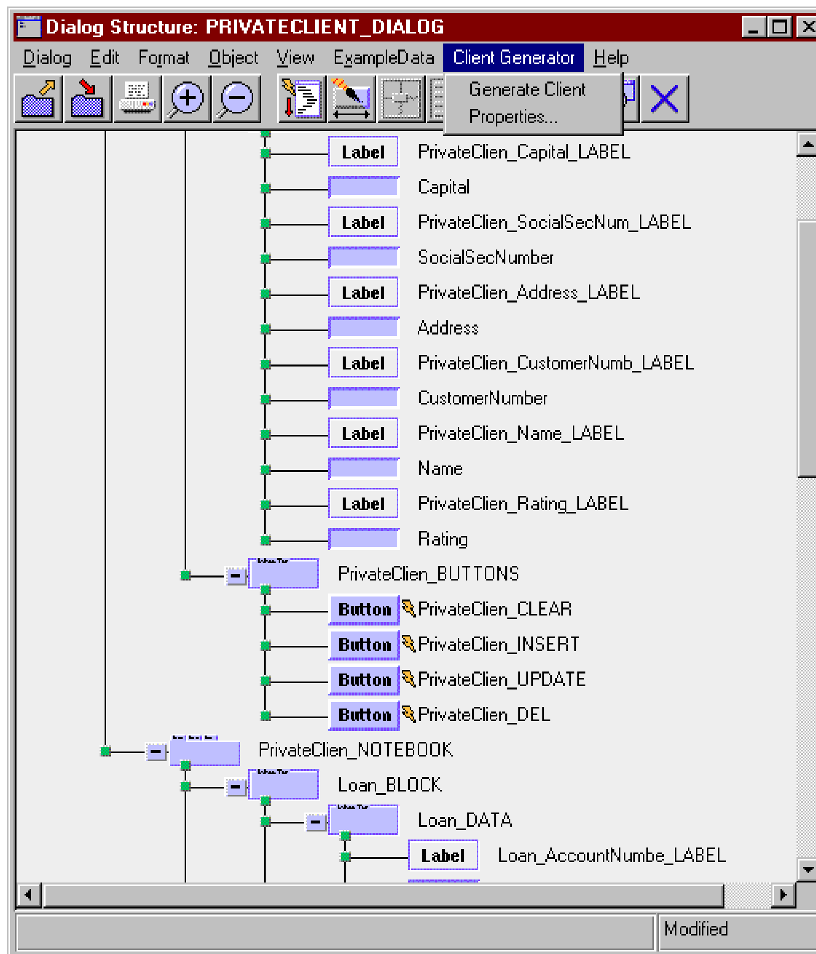

Figure 4.3 – The dialog designer tool

When the dialog model is sound, the final user interface is generated. This is done with the client generators. In Genova 6.1, it is possible to generate code to C++, java 1.1, Visual Basic and Motif. Genera generate much of the client's source code, in a three-tier model. It generates user interface definition files, controller instantiaters and call-back code skeletons. This enables the developer to quickly build the client part of the system. The code for implementing logic must be programmed by hand.

4.1.2 Implementing the case using Genova

The model of the case presented in chapter two was developed using the COMDEF UML profile. This profile consists of stereotypes such as entity, service, userservice and event [Kvalheim, 1999]. The services provided handle the entities, and the user service is the client's link to the underlying DCP. (COMDEF is described in detail in chapter 5.)

When using Genova to make an object selection that reflects a use case, it has to look at the entities, and not the methods of the user service. The object selection is done by viewing the use cases and selecting the entities that are used in a particular use case. As an example, the case in chapter two has a use case that describes the process of registering a new customer. The user service reflects this use case by offering a method called "newCustomer()". Because geneova do not understand the architecture of COMDEF and hence the consept of a userservice. This usecase has to be modeled by knowing the usecase and base it on the entity "CUSTOMER".

When the object selection is complete for all usecases, then dialog designer tool is used to make a model of the user interface. Every use case is transformed into a dialog model of the user interface. The developer can then modify the dialog model to his or her satisfaction. When this activity is finnsihed code genration to the target platform may be preformed. For test purposes, java was choosen as the target platform, when the testcase was imlpemented with the use of Genova. The linking between the client and user service had to be programmed by hand, as well as all application logic.

4.1.3 Evaluation of Genova

Genova is not intended for COMDEF and do not understand the concepts defined by this framework. It is a tool designed for tree tier architecture and provides code generation of the database, the client with the user interfaces and the glue between them. However, when used with COMDEF the only useable part is the user interface definitions. Genova generates user interfaces with some generic code, such as event handling of the different interface constructs (buttons, text fields etc), but all client logic must be programmed by hand.

The next sections will evaluate Genova with respect to the requirements in chapter three.

4.1.3.1 Productivity

The table below shows the productivity requirements defined in chapter three and shows how well Genova perform in regard to these requirements.

Table 4.1 – simplicity requirements

Requirement	**Supported**
P1 - Code generate the generic code	1
P2 - Code generate user interface	2
P3 - Code generate from graphical notation	2
P4 - Model environment independent	0
P5 - Graphic editable model of GUI	2
P6 - Template support	2
P7 - Developer control	2
P8 - Support for third party gui components	1
P9 - Support for interface guidelines	2

The generators of Genova generate some of the generic code. All event handlers are developed as skeletons that must be implemented by the developer. As stated, there is no connection to the underling DCP through the user service, because Genova does not understand the COMDEF concepts (P1).

The user interface is generated with the required functionality, and the code generation is from a graphical notation (P2, P3, and P5). The current version of Genova only supports a few UML modeling tools, which does not qualify as model environment independent (P4). The developer uses a modeling tool to modify the dialog model generated from the object selection, which satisfy the developer control requirement (P7). Support for interface guidelines is given through use of templates (P6, P9). The developer may create new templates, which in turn can obey a selected user interface guideline. Genova has support for ActiveX components, witch partially fulfill the support for third party user interface components requirement (P8).

4.1.4 Modeling concepts

In Genova, all development is model based. In addition, Genova utilizes UML models to generate the client. Table 4.3 shows the evaluation of Genova with respect to the modeling requirements.

Table 4.3 – Model Concept requirements

Requirement	Supported
M1 - Model based Framework	2
M2 - Understand and use the COMDEF metamodel/ UML profile	0
M3 – Declarative models	1
M4 - User interface and application logic in separate tiers	1

As stated, all development in Genova is model based, apart form the implementation of application logic in the generated client (M1). It does not understand the different concepts of the COMDEF framework, and cannot directly generate clients to this framework (M2). The object selection is done form a domain model of the underlying system, and the dialog model represent the user interface, but there is no model that fully describe the information flow between the underlying system, and the client, nor the order of events of the client (M3). Genova's architecture proposes a distinct separation of the UI-layout, UI functionality, business logic and database issues (M4).

4.1.5 Multi platform support

Genova has support for a selected range of target platforms. The user of Genova cannot add new platforms to the framework. In the Genova 6.1 release, the selected targets are of same thickness (Java, C++, Motif, and Visual Basic). There is no support for targets such as mobile phones or web- applications.

Requirement	Support
MPS1 – Multi platform support	1

4.1.6 Evolution

If system evolution forces changes, then this is handled as a new iteration of the development process. Small changes to layout can easily be managed by editing the dialog model, and reusing the hand implemented application logic. Orthogonal changes are managed in the same way. However, if there are big changes to the model then this may require that all hand-implemented code must be reprogrammed. The table below shows the evaluation of the evolution requirement.

Requirement	Support
E1 – Support changes in an easy manner	1

4.2 Mobi-d Overview

Mobi-d is a model based interface development environment, developed at Stanford University [Puerta, 1997]. It focuses on user-centred development, in an iterative development process. Mobi-d uses declarative models to represent all relative aspects of the user interface. This evaluation of Mobi-d is based on the reading of published articles on Mobi-d [Puerta, 1997] [Puerta, Einstein 1999] [Puerta, 1998] [Puerta, Cheng, 1999]. The

papers published on Mobi-d does not reveal how the different user interfaces are generated, nor do the say anything about how much of the source code that is being generated, but they describe the methodology, as well as the development cycle to be used with Mobi-d. Mobi-d is evaluated as a representative for the different model based interface development environments that uses declarative models to capsulate the different aspects of the user interface. This group of user interface tools has been labelled MB-IDEs, which is a short for Model Based interface development environment. The tool has not been tested, because it was not available during the work on this thesis.

Mobi-d uses a componential approach, which means that every widget or interactor is either an ActiveX control or a Java applet. All widgets are considered black boxes by Mobi-d, to avoid low-level modeling.

4.2.1 Mobi-d work protocol

Figure 4.4 depicts the development cycle that is used with Mobi-d.

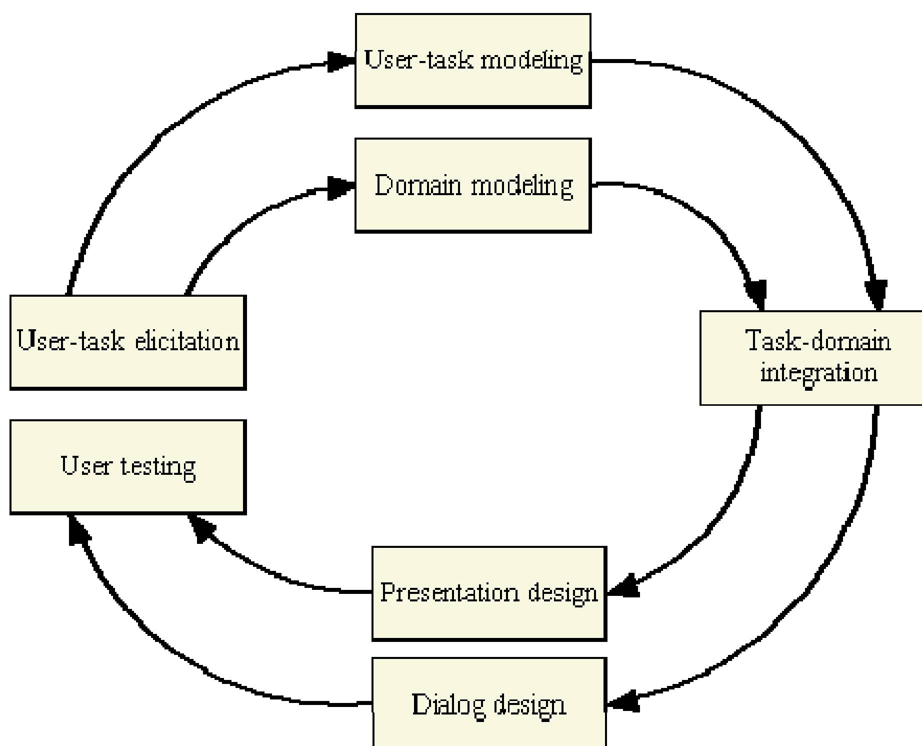

Figure 4.4 – Mobi-d development cycle

As stated, the development cycle of Mobi-d is an iterative process. It starts by describing the user tasks in an informal manner. The goal of this phase is to get an overview of the user tasks that the application shall provide user interfaces for. The sketching of the user task is done in a tool provided by Mobi-d, and the output of this task is used as input in the next development phase. Besides the technical goal of sketching the user task, another goal is to establish communication between the developer and the end user. This first development identifies the different tasks and objects that must participate in the task model.

The next step is to construct the user task model and the domain model. A skeleton task model is imported form the first development phase, and this model is refined by setting properties of the different tasks, and objects identified in the first phase. This step will

|produce a formal model of the tasks, and domain objects that is the foundation of the application, and hence the applications user interfaces.
The next step in the development cycle is to integrate the user-task and domain model. This is done by associating the domain objects with specific tasks. When a domain object is associated with a certain task, a relationship between them is constructed.

When the domain model and the task model are fully integrated, the next step is to produce a dialog model and a presentation model. These two models are constructed in parallel. The dialog model represents the human computer interaction, while the presentation model represents how the interaction components of the dialog model appear in the different dialog states. The presentation model is a hierarchical decomposition of the possible screen displays into groups of interaction objects, from the dialog model.

The last step of the development cycle is to produce and test the user interface. The evaluation of the produced user interface may require that a new iteration of the development cycle is executed. All steps of the development cycle are highly interactive in Mobi-d. This is done by providing tools that support all the activities within the Mobi-d development cycle.

4.2.2 Implementing the case with Mobi-d

This section describes how the case of chapter two could be implemented, using Mobi-d. As stated, tool has not been tested, and the evaluation is done by reading published articles about Mobi-d.

The first step is to sketch the different task. When developing a client for COMDEF, all the tasks of the user interfaces are documented through the use case models, so this first step might be skipped, or it could be used as a first step to establish the use cases. However, the methodology that is to be used with COMDEF produces the use cases in a different way. The work on developing the user-task, and the domain model should be straight forward, considering that all entities and use cases are already produced. After that, the next step is to produce the dialog and presentation model. This would probably make the integration of the two models easier as well, because the use cases have already described which domain objects that is required for the different tasks. The use cases implicitly reveal the structure of the presentation model. When both the dialog model and the presentation model are present, the user interface can be generated.

Unfortunately, the articles do not describe the process of generating the user interface. The deliverables from Mobi-d is the user interface, and the application logic must be programmed by hand. This means that all COMDEF relevant aspects must be programmed by hand, together with the application logic

4.2.3 Evaluation of Mobi-d

In the following sections, Mobi-d will be evaluated against the requirements of chapter three.

4.2.3.1 Productivity

Mobi-d focuses on generating the user interface from declarative models. The advantages of the Mobi-d approach are user-centered development and that the modeling of the user tasks is moved from the head of the developer to declarative models in the tool. Mobi-d also provides guidance in the development process. Table 4.6 evaluates Mobi-d against the productivity requirements of chapter 3.

Table 4.6 – Productivity requirements

Requirement	**Supported**
P1 – Code generate the generic code	1
P2 – Code generate user interface	2
P3 – Code generate from graphical notation	2
P4 – Model environment independent	0
P5 – Graphic editable model of GUI	2
P6 – Template Support	2
P7 – Developer control	2
P8 - Support for third party user interface object	2
P9 – Support for interface guidelines	2

From the articles it not clear how much of the generic code that is developed. However, the deliverables from Mobi-d is the user interface and it is assumable that it delivers the same kind of interface as other interface builders. That is the user interface and skeleton code of the methods of the different widgets of the user interface. This qualifies the P1 requirement partially

The user interface is totally code generated, and it is code generated from a graphical notation. (P2, P3).

Mobi-d does not use any UML modeling tool as basis for generating the user interface, so the P4 requirement is not fulfilled.

Mobi-d lets you edit the user interface in a graphical tool (P5) and all stages of the development process are interactive, allowing developer control (P7). All widgets are either java applets or ActiveX controls, which fully support the requirement (P8). The Mobi-d architecture has a knowledgebase for style guides and interface guidelines that are incorporated in the development process (P9). Complete interface designs can be stored as declarative models which may serve as templates for later projects (P6):

4.2.3.2 Modeling concepts

There is little support for modeling concepts in the context described in chapter three. The table 4.7 evaluates the modeling concept requirements of chapter 3.

Table 4.2 – Modeling concepts

Requirement	**Supported**
M1- Model based framework	1
M2 – Understand and use the COMDEF metamodel /UML profile	0
M3 – Declarative models	2
M4 – User interface and application logic in separate tiers	1

Mobi-d is model based in the sense that it describes all relevant parts of the user interface as declarative models (M1, M3). It does not get any input from a UML model; hence, it does not understand the concepts of COMDEF (M2). The separation of user interface and the application logic is unclear, but it is assumable that it is possible to separate them into tiers, because the application logic must be programmed by hand (M4).

4.2.3.3 Multi platform support

Since the deliverables of Mobi-d are java code, it may be used on several platforms. It is however not possible to generate source code to platforms of different thickness. This does not qualify as multi platform support.

Table 4.7 – Multi platform support

Requirement	Supported
MPS1 – Multi platform support	0

4.2.3.4 Evolution

As with Genova the simple evolution changes may be done by doing one or more iterations of the development cycle. Orthogonal changes may be done in the same way. If there are major changes to the user interface, then this might require big changes to the application logic, as well as the user interface. Mobi-d partially fulfill the evolution requirement

Table 4.8 - Evolution requirements

Requirement	Supported
E1 – Support changes in an easy manner	1

4.3 Interface builders overview

Interface builders are part of many programming applications. The provide a graphical user interface where user interfaces can be built using a palette of different user interface objects, and many provide templates that can be chosen to build different types of standard user interfaces, such as a logon – or splash screen. The interface builders generate code at design time in the language that the programming environment supports. A well-used method is to generate comments in the source code that is used by the interface builder to visualize the interface at design time. This makes the source code difficult to edit, because besides knowing the programming language the developer also must know the semantic of the different comments that are generated. As an example of such an environment, the case was implemented using Visual Café [Symantec, webgain, 2005], which is a programming environment for java, by Symantec. Interface builders are an alternative to be considered because it provides some automatization to process of building the client, by generating the source code for the user interface

4.3.1 Visual Café work protocol

Visual café is a general java-programming environment. It provides templates for different forms of java applications, such as java beans, applets and standalone applications as well as providing a user interface builder. Visual Café is a typical programming environment and does not offer any particular work protocol.

4.3.2 Implementing the case using Visual Café

Implementing the client in Visual Café requires that the developer fully understand the COMDEF architecture, and that the developer knows the layout he or she wants. The user interface builder lifts the abstraction level form a textual to a graphical environment, but the user interface still has to be developed from scratch. When the all user interfaces of the application are built, all application logic has to be programmed by hand. Because the client also needs to import the userservice packages, some reconfiguration of the environment is needed, in order to be able to debug and compile the application in Visual Café. Beside the user interface the different events such as mouse click and keyboard events are also generated, but the logic has to be implemented by hand.

4.3.3 Evaluation of Visual Café

Because Visual Café is a java environment, the only platforms that are supported are those who support java. Therefore, if the client is to be supported on other platforms, other developer environments have to be used. The only automatization that is offered by these tools is the automatic generation of the user interface and the events that handle user interaction. This implies a big effort by the developer. In the following Visual Café is evaluated against the requirements defined in chapter 3. This evaluation can also be viewed as a general evaluation of user interface builders because most of them offer the same functionality.

4.3.3.1 Productivity

The table below shows the productivity requirements defined in chapter tree. And shows how well Visual Café performs in regard to these requirements.

Table 4.1 – Productivity requirements

Requirement	Supported
P1 - Code generate the generic code	1
P2 - Code generate user interface	1
P3 - Code generate from graphical notation	2
P4 - Model environment independent	0
P5 - Graphic editable model of GUI	2
P6 - Template support	1
P7 - Developer control	2
P8 - Support for third party gui components	2
P9 - Support for interface guidelines	1

Event handling is generated by Visual Café when the user interface builder is used. The source code for the user interface is also generated, but the user interface has to be modeled from scratch. So, the process of developing the user interface is not automized, with the exception of the templates that are offered. The templates generate some very basic methods such as a basic menu, about – and quit dialogs. The connection to the user service and the DCP are not handled automatic by the code generation Visual Café offer. Therefore, the requirements P1, P2 and P6 are only partially satisfied. The code generation that does take place is based on the modeling of the user interface in the graphical user interface builder, which also lets you edit the user interface, which satisfies requirements P3 and P5. P4 is not supported because Visual Café is not integrated with any UML modeling tools. P7 and S8 are fully supported implicitly since Visual Café is a programming environment where every programming is done by hand. Interface guidelines are also implicitly supported because the user interface has to be modeled by hand.

4.3.4 Modeling concepts

Table 4.3 – Model Concept requirements

Requirement	Supported
M1 - Model based Framework	0
M2 - Understand and use the COMDEF metamodel/ UML profile	0
M3 – Declarative models	0
M4 - User interface and application logic in separate tiers	1

None of the modeling concepts requirements are supported by Visual Café, except M4 which can be supported by separating the tiers by design during the implementation.

4.3.5 Multi platform support

User interface builder such as Visual Café are developed for a specific programming language, and therefore by nature only supports the platform for which it is intended.

Requirement	Support
MPS1 – Multi platform support	0

4.3.6 Evolution

Changes are done in the same manner as the first development. The chosen architecture may be designed to support changes, but Visual Café does not have any special functionality to support changes.

Requirement	Support
E1 – Support changes in an easy manner	1

4.4 Summary

This chapter has looked at some of the existing solution to the case presented in chapter two. Some different technologies have been applied to the case and the result has been evaluated against the requirements of chapter tree.

Chapter 5

Multidev's technology relationships

This chapter introduces different software technologies that either is used in the development of Multidev or is used by Multidev in the development of COMDEF clients.

5.1 COMDEF

The Component Development Framework (COMDEF) described in [Kvalheim, 1999] is a framework for developing component based distributed information systems. The framework was developed in the context of the Open Business Object Environment (OBOE) Espirit project [Oboe, 1999] in which SINTEF was a project partner.

While COMDEF focuses on rapid development of the backbone of component based information systems, Multidev focuses on generating clients that can run on COMDEF systems.

The framework consists of many different parts, which are presented in detail in [Kvalheim, 1999], but an overview of COMDEF is presented here in order to understand how Multidev uses different parts of COMDEF as well as how Multidev fits into this framework.

The main parts COMDEF are:

- **Metamodel /UML Profile** – This metamodel is based on the UML metamodel and introduces meta-classes that are to be used in system modeling with COMDEF.
- **CML –** Component Modeling Language is a language for lexical representation of the concepts of COMDEF. The CML language and the COMDEF metamodel maps one-to-one, which makes it possible to describe the system in a graphical or a lexical notation.
- **Mappings** – COMDEF provides different code generation tools that are called emitters. These tools use the CML representation of the system to generate source code to different architectures. Emitters can also be used to generate database schemas to make the system persistent.

When a system is to be developed with COMDEF a methodology that is capable of producing a UML model must be used. The methodology is not a part of COMDEF, but uses COMDEF as a development tool.

As stated, COMDEF provides a metamodel (UML profile) that is to be used when modeling a COMDEF system. This profile includes stereotypes of the different elements of the COMDEF framework. The reason for developing the metamodel was to be able to describe the different parts of the COMDEF framework with more precision. This will ease the mapping to CML as well as the code generation.

When the developer has modeled the system, the model is converted to a CML representation. The CML language was developed in order to close the gap that the COMDEF developers found between the graphical and lexical modeling languages. The CML language is capable of mapping the graphical COMDEF model one-to-one, making it possible to transform the model from a graphical notation to a lexical notation and vice

versa. The CML language is based on the Interface Description Language (IDL), which was defined by OMG in the CORBA 2.0 specification [CORBA, 1996] and the Object Definition Language (ODL), defined by ODMG in ODMG 2.0 standard [ODMG, 1997]. The CML representation is generated and this is used by the different emitters to generate source code for the selected architecture (CORBA, EJB, and COM).

The conceptual architecture of COMDEF figure 5.1 shows the main modeling concepts of COMDEF.

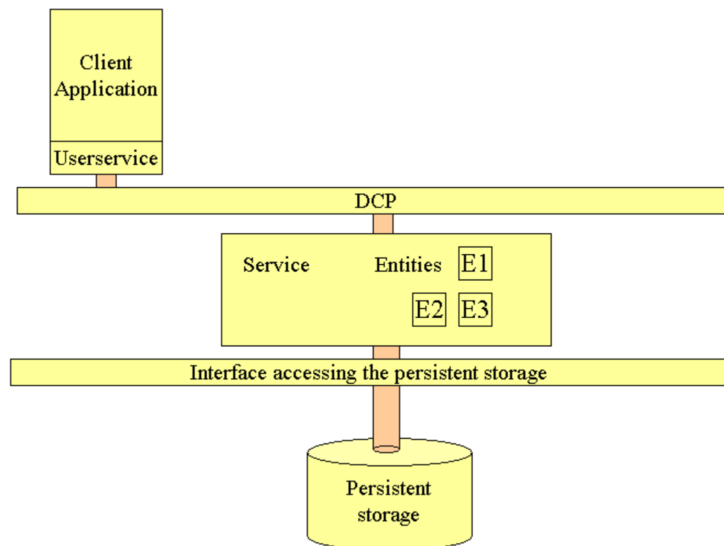

Figure 5.1 – The COMDEF conceptual architecture

These are the userservices, services and entities. As the figure shows the client application accesses the services trough the userservice and the services encapsulate the entities.

5.1.1 The user service

The user service is, as the figure shows, the only part of the client that is generated by COMDEF and the user service provides the different operations of the services to the client application. Depending on how much information that is available at modeling time the userservice can be fully or partially code generated. There are two main scenarios that implies how much of the user service that can be code generated. The most complete solution is if the service and the userservice are mapped one-to-one. In this case, the user service can be fully code generated, and is modeled in COMDEF by stating that a userservice *provides* a service. The other scenario is when the services are known to the userservice, but how they are used is not known. In this scenario, the only thing that can be code generated is an abstract implementation of the userservice. This scenario is modeled in COMDEF by stating that a userservice *requires* a service to perform its job and COMDEF can code generate the connection to the right services, but most of the implementation must be done by hand.

5.1.2 The services

The services are the components from a COMDEF viewpoint. A service is instantiable, stateless and access transparent. The meaning of access transparent in this context is that a service is registered and available on the net. There can only be one instance of a given service at a given time. The reason for this is that that the user service should be able to connect to the service through code generated code.

A service will offer a set of operations to client applications, and the operations provided are not limited to one application, but a service may offer a range of operations that cover the needs of a business domain. The user service, however, is application specific and offer only those methods from one or more services that are needed by a specific client application. As a special case, this may include all the operations of a service.

5.1.3 The entities

The entities are the data objects in the COMDEF framework and are logically contained in services; hence, the entities are intended to be accessed through services. Entities may have relationships with other entities and with services

5.1.4 Events

The COMDEF framework includes an event model that is based on the event model introduced with Java 1.1 [Flanagan, 1997]. In the COMDEF event model, both a service and an entity may be both an event source and an event listener. The event source has a list of all event listeners that should be notified of an event. The event listener must register it self to event source in order to be notified of an event.

5.1.5 Mapping

The COMDEF framework lets the developer map the system to a selected range of middleware technologies, such as CORBA 2.0, EJB and COM, making COMDEF a powerful development framework. CORBA is the only platform that has been vastly tested during the development of COMDEF.

5.1.6 Post COMDEF

While working on this thesis model based development has moved forward. The most significant approaches are OMG MDA and Domain Specific Languages from Microsoft. (Both of these are presented in more detail in Chapter 10)
COMET [comet, 2006] is a Component and Model-based development methodology. Development on COMDEF has stopped, but COMET can be viewed as the next generation of COMDEF. In addition to the framework itself COMET provides a set of tools that can be used when developing through the methodology. A difference between COMDEF and COMET is that COMET does not use CML as bases for code generation. Instead code generation is done from a graphical notation (UML) through Platform Independent (PIM) and Platform Specific Models (PSM) (PIM and PSM are both described in more detail in chapter 10).

5.2 XML

The extensible markup language (XML) is a language that is derived from the *Standard Generalized Markup Language* (SGML). XML is a standard of the World Wide Web consortium (W3C) and the first release of XML came in 1998 [W3C-XML, 1998].

XML is a method of putting structured data into a text file, in a way that is easily readable for computers. XML consists of two parts: the XML document and a Document Type Definition file (DTD). The DTD describes the language by providing the different tags that are legal in the XML language for which it is intended. The XML document contains some kind of information that is structured by the tags defined in the DTD of that instance of XML [Maruyama et al., 1999].

5.3 XMI

The Object Management Group (OMG) has defined a standard for describing metadata in XML based languages. This standard has been named XML Metadata Interchange (XMI)

[OMG XMI, 1999]. The XMI standard provides a set of rules for generating a XML DTD from a metamodel so that the metamodel can be described in a XML based language. Although XMI was created to describe metadata, it can also be used to describe instances of the metadata, such as UML models. In order for XMI based languages to be suitable for exchanging information between different products some DTD standards have been defined. One of them is UML XMI DTD, which makes it possible to describe UML instances as a XML document. All products that support this standard now have the ability to share the UML model between them [Frankel, 2001].

5.4 XSLT

The XSLT is a transformation language by W3C [W3C-XSLT, 1999], which is intended for transforming XML documents into XML documents. According to the W3C, XLST is a part of XSL, which is a language for defining formatting and presentation of XML documents [EC3-XSL, 2001]. Besides XSLT, the XSL language consists of XPath and XSL Formatting Objects (XLS FO). The XPath is better considered a sub language of XSLT for selecting specific elements of a XML document while XSL FO is used to define the formatting of the XML document.

The process of transforming the XML document is done by writing a XSLT style sheet that describes what output that should occur when a certain pattern in the input XML file is matched. The XSLT style sheet and the XML file are used as input to a XSLT processor that generates the output.

As stated, XSLT was invented for transforming XML documents into XML documents. However, XSLT is not limited to that. The output from the processor could be any kind of format, such as an html file or plain text.

5.5 UIML

UIML stands for User Interface Markup Language and is a XML based language for describing user interfaces. UIML is a product developed by Harmonia [Harmonia, 2001] and the Center for Human-Computer Interaction at Virginia Tech [Marc, 2001].

The idea behind UIML is that a user interface is described in UIML once and this description can be used no matter what kind of hardware or software you will use to run your application. The user interface is interpreted by different renderers that are software and hardware specific. At run time, the UIML file is interpreted and displayed on the selected platform. UIML describes the user interface in two parts [UIML 1.0, 1997]. Appliance independent user interface definition is defined separately from appliance dependent style sheet that guide the placement and appearance of user interface elements. With the separation a user interface definition can be used on any platform for which there exists a style sheet. A UIML document describes the appliance independent user interface definition is done by defining the user interface with generic tags. Instead of defining platform specific widgets such as windows or menu items, they are defined as *parts*. These parts are given a name and attributes that together should be enough to map the specific part into a platform specific user interface widget.

As stated, UIML is a XML language that is defined by a DTD. The DTD specify tags that are valid in UIML. These tags are generic tags that lift the abstraction level from being specific about windows, buttons and other widgets to meta user interface parts.

UIML is now in its third revision. The UIML 3.0 version differs from the 2.0 version in the following ways [Abrams, Helms 2004]:

1. Support for dynamic user interfaces:
   A user interface is viewed as a virtual tree, whose initial content is specified by the <structure> element, and which can change during the lifetime of the user interface.

2. Support for multimodal user interfaces:
   A user interface can contain multiple <interface> elements that are simultaneously used and kept synchronized.

3. Refinement of many language constructs, based on implementation experience with UIML 2.

Although other technologies have emerged that can compete with UIML in the context of Multidev, UIML still has features that make the technology compelling for the Multidev framework intentions. Inn addition to achieving more matureness in its third release, the number of renderers supporting the language has grown in the last years.

Today, UIML is being standardized by OASIS. [Oasis 2006] It is till a goal for the UIML developers to make UIML an open language that is freely available to developer around the world. [Abrams, Helms, 2004].

5.6 Java

Java is an object oriented programming language, developed by Sun Microsystems [Sun, 2006]. Java has a syntax that is similar to C++, but Java has left out features such as overloading, pointers and multiple inheritances because the developers of Java found these features error prone. Java is an interpreted language that is platform independent. It gains its platform independency by interpreting byte code on a virtual machine called Java Virtual Machine (JVM) that is ported to different platforms. Because Java is interpreted it is slower than native languages, such as C++.

Multidev is not intended to be a java specific tool, but is meant to utilize UIML to produce user interfaces for a range of platforms. However the current version is limited to producing Java interfaces. Java is actively being developed and used all over the world.

In Multidev, Java has been used on two levels. Firstly Java was the chosen language for the COMDEF tools use by the Multidev framework. In addition the java was used as a target platform for the generated clients.

5.7 UML

The Unified Modeling Language (UML) is a non-proprietary, general-purpose modeling language. UML can be used to specify, visualize, construct, and document all artifacts of a software system. Through the use of stereo types, developers can extend the language to include new concepts.
UML has received great acceptance since first submitted to the OMG in 1997 [Wikipedia, UML 2006]

The latest release of Unified Modeling Language UML is the 2.0 release. In-depth information about UML 2.0 can be found at [UML 2005 #1]. A summary of what is new in the 2.0 specification [UML 2005 #2]:

- Nested classifiers
- Improved Behavioral modeling
- Improved relationship between structural and behavioral models

5.8 Summary

This chapter has presented modern software technology. The technologies presented are either used in the development of Multidev, or is used in Multidev to develop clients to COMDEF based systems.

Chapter 6

Multidev

Multidev is a framework for developing clients to COMDEF based systems. The key point of this framework is to utilize the information in the UML model of a COMDEF system, in order to generate as much of the clients source code as possible. The generated clients should be able to run on different platforms. The challenges are to generate the various user interfaces that the client need to present and retrieve information from the end user, as well as the information flow between the user interface, the application and the distributed environment.

This chapter introduces the concepts of Multidev. An overview of the development process is given ass well as a description of the architecture and the extensions to the COMDEF framework, and how these extensions are used.

6.1 Limitations and advantages

The whole framework has not been implemented in the existing version of the tools, due to time limitations. The implemented part serves as a proof of concept rather then a ready to be used developing framework.
The clients that can be generated by this framework are not capable of expressing the user interfaces with all sorts of fancy graphics, but are meant to support building interfaces for distributed computer systems that give and request information as text and or simple graphics. However, the proposed framework lets the user add third party components, or new widgets, to the user interface, which can be used to produce more complex user interfaces. The advantages of Multidev, is that it has a well-defined client architecture, that it makes it easier to develop the client through code generation and that by defining the client once, it is possible to map it to different platforms.

6.2 Multidev overview

Multidev is an add-on to the COMDEF framework, but the ideas may be used in other environments that build on the same principles as COMDEF. In the development phase of a distributed system Multidev may be used to generate prototypes and the prototype will then evolve together with the rest of the system, coming closer to the finished product for each iteration. Figure 6.1 shows an overview of the development process with MULTIDEV.

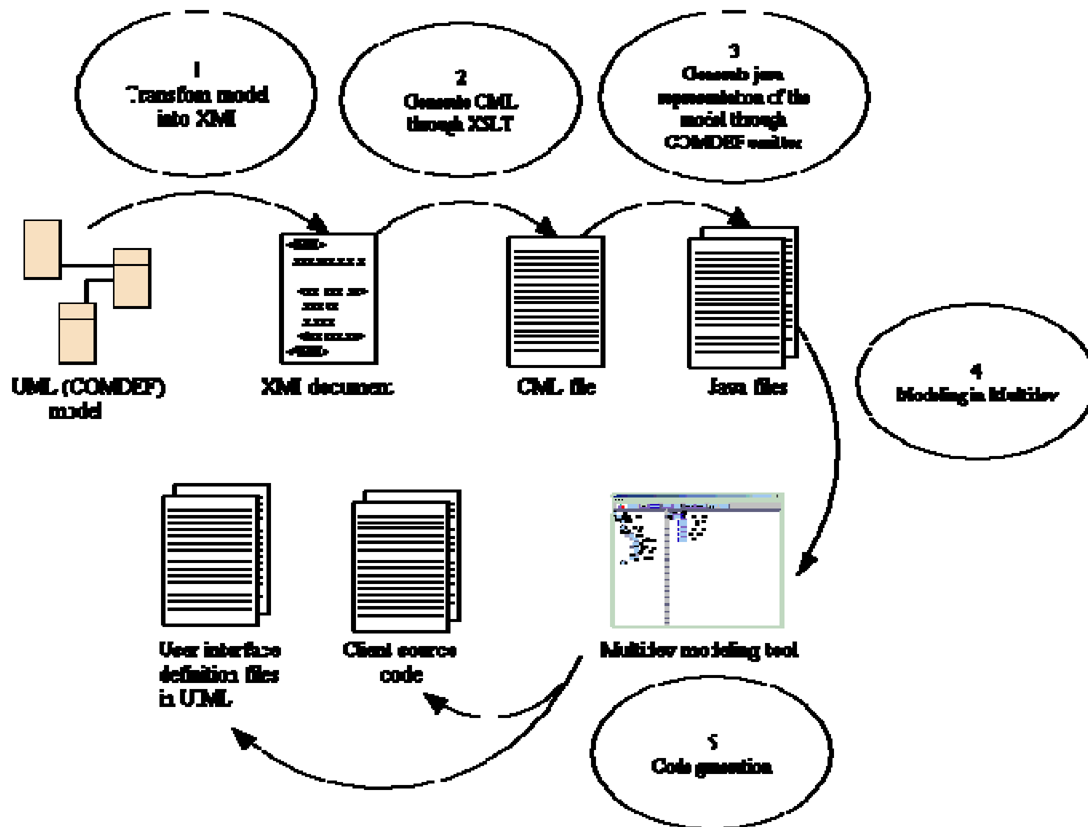

Figure 6.1 – Multidev development process overview

As an add-on to COMDEF, Multidev follow the same development process as COMDEF. When using COMDEF a development methodology that produces a UML model of the system must be used. The methodology is not a part of COMDEF nor is it a part of Multidev. As described in chapter 5, the OBOE project proposes a development methodology that is well suited for development with COMDEF and hence Multidev. This development methodology shares the properties of the methodology that is suggested by the developers of UML [Booch et al., 1999], in that the development process (methodology) should be:

- Architecture-centric
- Use case driven
- Iterative
- Incremental

The methodology proposed by Oboe respects these properties in the following way:

Architecture-centric
Architecture-centric will be supported implicit when modeling systems with the COMDEF framework in mind because the core of COMDEF is the architecture presented in chapter 5. The same apply for Multidev, who uses the same architecture, but adds a layer to the client part, by separating the application logic and the user interface. This will be described later in this chapter.

Use case driven
The use of use cases has proven to be a useful tool to unveil a system under development. Use cases have many different angels depending on what information the developer wants to specify. One angle that has proved itself efficient is to use them as a way of structuring the requirements of a new information system [Cockburn, 1997]. The methodology proposed by OBOE exploits use cases to structure requirements. Besides structuring the requirements, the use cases will describe the different tasks that client application must handle through the user interfaces, which is vital information to the task model. (The task model will be described in chapter 7).

Iterative and incremental
For simple software systems, it might seem feasible to sequentially define the problem, design the solution, implement it and then test it. For complex system, this approach is not feasible. The complexity of modern computer systems makes it difficult to produce a good solution, or even define the problem in one loop. An iterative and incremental development process makes it easier to produce good software through refinement and incremental growth. An iteration should result in either an internal or external release that can be tested and evaluated. The incremental steps add new functionality to the system, and thereby moving it towards the final release. Multidev can be used to produce a prototype for testing purposes.

6.2.1 The Multidev process

In the early faces of development, the first iterations will produce the first UML models, and Multidev can be used to produce prototypes of the system. Figure 6.1 show the development process when using Multidev. The figure shows the different steps from the COMDEF UML model to the client generation. The first steps are technical steps to read the COMDEF UML model into the Multidev modeling tool. The UML modeling tool used to model the system must either support exportation to XMI or CML. In the current version of Multidev, the model is first exported into XMI with the use of the UML DTD. Then a XSLT script is used to convert the model into CML. This process will be elaborated more in section 6.4.

A long side the domain model (original COMDEF model), Multidev introduces a new view of the system. This view contains a skeleton of the user interface layout. Both the domain model and the dialog layout model are considered as the COMDEF model and both are contained in the same CML file. The CML file is used by a COMDEF emitter to produce two java classes that holds the model as java objects. This is done by parsing the CML code and then generating the java classes. The Multidev modeling tool uses these classes for dialog modeling.

As stated the emitter produces two different classes; one that represent the domain model, and one that holds the skeleton layout of the dialogs. These two models constitute the entry point of the modeling effort in Multidev; and hence must be available before any work in Multidev can begin. When the first three steps in figure 6.1 are complete, the user may start the task of modeling the user interfaces in Multidev. The first task of this activity is to model the task-model that describes the functionality and the information flow of the client. After that, the task model is mapped to a presentation model that represents the various user interfaces of the client. When both models are complete, Multidev will code generate the client to the selected platform. Multidev will generate UIML files that constitute the user interface and the application logic will be code generated into the selected language. Multidev must know what kind of platform the client will run one, because it uses two different strategies for displaying the interface, depending on the platform. This will be described in more detail in the next section

6.3 Architecture

Multidev adds a new layer to the COMDEF architecture by separating the client's application logic from the clients user interface. The architecture is depicted in figure 6.2.

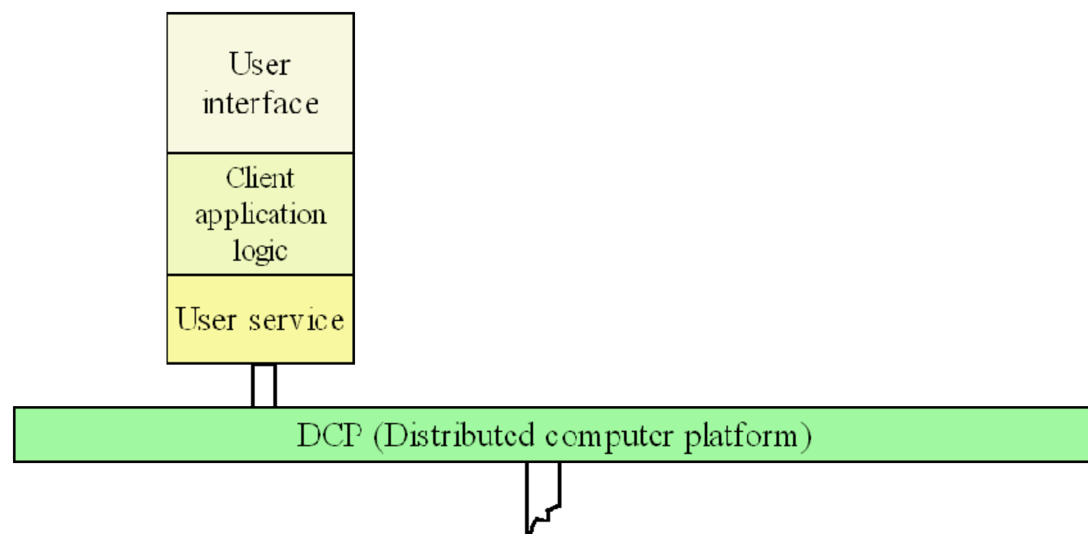

Figure 6.2 – Multidev conceptual architecture

The conceptual architecture lives on top of the COMDEF architecture with the user service as the bottom layer that connects to the services trough the underlying DCP. The user service provides the services to the client as methods that deliver and accept entities to and from the client as well as providing the service specific methods.

The middle tier is the client logic that handles the connection to the userservice and to the user interface. The user interface presented to the end user, generates events in response to user interaction. These events are caught and handled by the client's application logic, through a renderer, as described next.

The client application uses a renderer to read and display the UIML files as user interfaces, as well as reacting on events trigged by the end user, such as mouse clicks or typing on the keyboard. These events are passed down to the application logic. There are two ways the UIML file can be rendered into a user interface, as figure 6.3 and figure 6.4 shows.

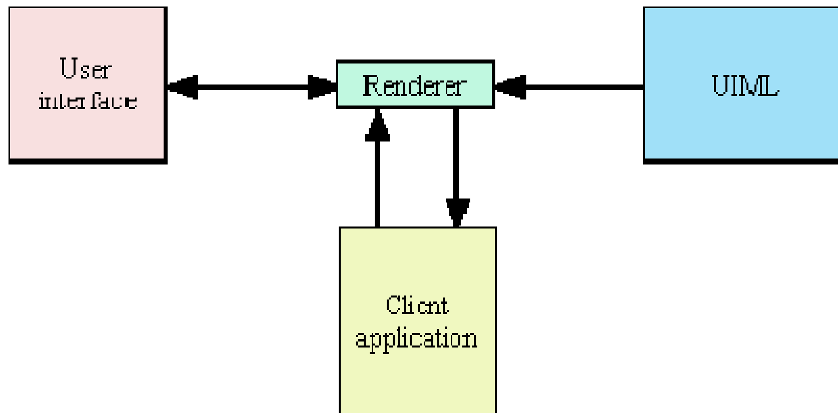

Figure 6.3 – Deployment of user interface through a renderer

In figure 6.3 the client application uses its own renderer to display the user interface. This method may be used if the application should be used as a stand-alone application implemented in java, C++ or other supported languages. The client application imports a renderer and tells it which UIML file to render. The renderer also catches the different events triggered by the end user and sends them down to the client application where they are handled.

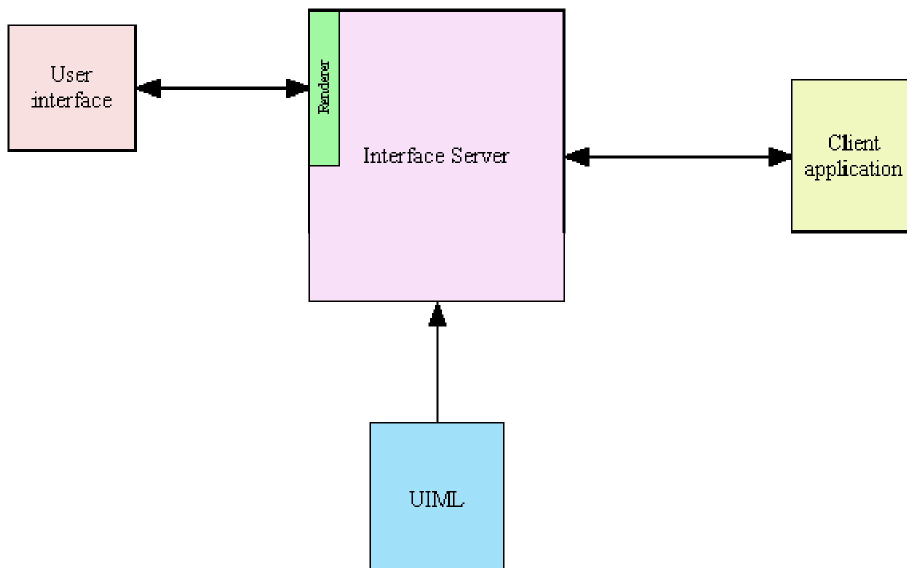

Figure 6.4 – deployment of user interface trough an interface server

In the case of figure 6.4, the target platform may be HTML or WML. In this case, the renderer lives on an interface server and converts the UIML document into the desired language. The interface server catches all events triggered by the end user and sends them to the client application.

6.4 XMI and CML

In the current version of Multidev, the UML model is exported into XMI. The reason for using XMI is to be independent of the UML modeling tool used. Both the XMI file and CML file contain the information needed as input to Multidev, but the CML file represents a subset of the information that is stored in the XMI file. In the development of Multidev Rational Rose was used as the modeling environment. Rose supports XMI through the UML-DTD. In addition to the model specific information, Rational Rose stores overhead information in the XMI file. The overhead information in this context is all information that is not relevant to the UML model, such as layout of the model elements, colors used, font properties and so on. This overhead information makes the XMI file very large. In the models used in the development of Multidev, the XMI file was on average 50 times larger then the CML file which both contain the information needed by Multidev.

Since both the XMI file and the CML file contain the same information, the XMI file could be used directly as input to Multidev. This strategy was tried by using the Simple API for XML (SAX) [Meggison -tech, 2000] for java to parse and extract the information from the XMI file. SAX was chosen because it is proven to be good with large documents that do not fit in memory and doing tasks on elements that are irrelevant to the surrounding document structure, such as extracting information of a specific element [Maruyama et al., 1999]. The SAX parser scans the XML file and generates events, such as the start and end of an element as well as the information contained in an element. By implementing an interface called *Documenthandler* that defines what should happen when the different events are triggered, the XMI file is parsed and the Multidev input files can be generated.

Although this approach is not difficult, it proved to be very time consuming to implement. The document handler needs to know both the structure of the XMI file and the tags defined in the UML DTD as well as the structure of COMDEF in order to be able to retrieve all information needed to build the input to Multidev. At SINTEF there is already developed an XSLT script to convert a XMI file to CML and the COMDEF framework provide tools to parse and generate code from CML. This strategy was chosen to save time. The XSLT script reads the XMI file and converts into a CML file. The conversion looses no information that is vital to Multidev. With the CML file, the existing technologies that are part of the COMDEF framework can be used to generate the java files containing the models, as described in section 6.2.1.

6.5 UML modeling

The COMDEF framework introduced a metamodel in order to allow a more precise description of the different concepts of the framework. Multidev introduces a new view of the system, which captures layout of the dialogs. The view represent the client view of the system being developed and is described by three new stereotypes. The changes are orthogonal to COMDEF meaning that the changes do not interfere with the existing COMDEF techniques. The new stereotypes represent an extension to the COMDEF metamodel. The extension is done through extending the UML metamodel in the same way as the COMDEF metamodel does, as described next.

UML provides four-leveled metamodeling architecture [OMG UML, 1999]. Table 6.1 shows the different layers of this architecture. The following section describes the relationships between the different models.

Table 6.1 - The different model levels

Model	Model example	Instances	Instance Examples

Metametamodel	UML meta-metamodel	Meta classes	Meta class, meta attribute
Metamodel	UML/Multidev meta model – UML/ COMDEF metamodel	Meta classes	Dialog, service, attribute
Model	Room booker model	Classes	Login dialog, Scheduler service, room
User objects (running system)	Room booker system	Objects	<Room_booker_login_dialog> <Rom_scheduler_service_32123> <Room_123>

The first level is the meta-metamodel. This model is responsible for defining a language for specifying a metamodel. The meta-metamodel is at the highest level of abstraction and only defines the building blocks of metamodels.

The next layer is the metamodel, which may be regarded as an instance of the meta-metamodel. The main task of the metamodel is to describe a language for specifying models. Examples of objects in this layer are classes, attributes and operations. The UML metamodel may be extended with new objects to provide a metamodel that makes it easier to describe the different building blocks of the domain that is being modeled. By giving the stereotype special properties, semantics and notation [Booch et al., 1999], the stereotypes become formal building blocks that are more specific to the modeling domain in question.

The next level is the model level. A model is an instance of the metamodel. On this level, the language of the metamodel is put to use to describe the system that is under development. The model layer defines a language to describe the information domain.

The last level is the user objects, or in other words, the running system. The running system is an instance of the model level, and consists of the objects such as an instance of the room object that now represent a physical room.

6.6 Extending the UML metamodel

As mentioned UML allows it's metamodel to be extended. There are different ways to extend the metamodel. The reason for these extension mechanisms is that some situations need additional features or notation beyond those defined by the concepts in the UML standard. The different ways to extend the UML metamodel are explained in the following.

- **Stereotypes –** The stereotype is the most important built in extension mechanism in UML. Any UML model element may be stereotyped and this feature provides a way to facilitate the addition of virtual UML meta classes with new meta attributes and semantics. This is useful when you want to introduce new things into the UML vocabulary to be able to model more precisely. A stereotype is rendered a name enclosed by guillemots and placed above the name of the element that is stereotyped.

- **Constraints –** Constraints makes it possible to add new semantics or change the rules that apply to a certain model element. Constraints can be written as free-from text or in Object Constraint Language (OCL) if the semantics of an element should be specified more precisely. The OCL constraints used in the conceptual metamodel of Multidev are only present in the metamodel to express the features of the new stereotypes in a

formal manner and are not enforced by the modeling tools on instances of the metamodel

- **Tagged value –** Tagged values are an association mechanism between a UML model element and some arbitrary information. Tagged values may be considered as metadata because its value applies to the element and not its instance.

6.7 Extending the COMDEF UML profile

Multidev introduces an extension that makes it possible to model the layout of the dialogs that together becomes the application or client. The extension makes it possible to keep the information about the client, and the dialogs it contains, together with the rest of the information that constitute the systems architecture in the object oriented modeling tool used. The extension may be viewed as a metamodel on its own, because it is orthogonal to the rest of the COMDEF metamodel. The conceptual metamodel (COMDEF extension) is depicted in the figure below.

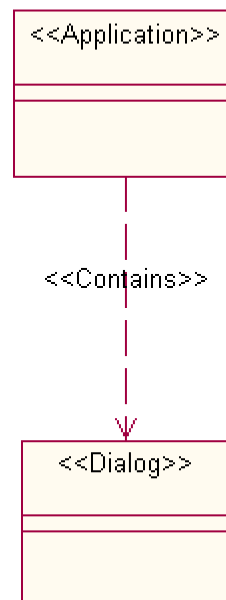

Figure 6.5 – Conceptual metamodel

The metamodel consists of three stereotypes as figure 6.5 shows. An application is a collection of different dialogs. The application is defined as a set of dialogs, or in other words, an application *contains* dialogs. The models that are instantiated from this metamodel are only meant to represent the structure of the application and is used as an input to Multidev. Neither *Application* nor *Dialog* should contain any methods or attributes, and this is expressed in the metamodel by OCL constraints. The relationship between them is also defined by OCL constraints. The OCL constraints are only present in the metamodel to formalize the elements and are not used to restrict the use of the elements during modeling of the system. The reason the OCL constraints are not used to restrict the use of the elements at modeling time is that the UML modeling tool used with Multidev does not support it. In the following, the different modeling elements will be presented.

6.7.1 Application

The application stereotype represents the client and may have one or more dialogs. The stereotype may not have attributes or methods. This is enforced by the following OCL constraint:

```
self.feature->isEmpty
```

In this expression, *self* is the application element.

6.7.2 Contains

The stereotype contains is a subclass of dependency from UML core. This dependency is used to illustrate that an application contain dialogs. The *contains* dependency must have an application element as its client and a dialog element as a supplier meaning that the client (application) is dependent on the supplier (dialog). This is expressed with the following OCL expression:

```
self.client.oclIsTypeOf(application) and
self.supplier.oclIsTypeOf(dialog)
```

Self is the *contains* dependency.

6.7.3 Dialog

The dialog stereotype represents a typical user interface dialog such as a login or registration dialog. The dialog stereotype may not have attributes or methods which is enforced by the following OCL expression:

```
self.feature->isEmpty
```

In this expression, *self* is the dialog stereotype

6.7.4 Extending CML

To be to use the COMDEF code generation tools the new stereotypes must be defined in the CML language. The CML language is based on the Interface Description Language (IDL) and the Object definition language. Multidev extends the CML language with the following:

- Application
- Dialog
- Contains dependency

The extensions to the CML grammar are presented in the following sections with examples. The keywords are shown in bold font while the non-terminals are shown in regular font. Neither the IDL- nor the existing CML productions are elaborated here. The complete CML grammar is given in appendix C.

6.7.5 Application CML grammar extension

The application cannot have any attributes or methods. It is only allowed to have one ore more dependencies to dialog elements. This described by the following grammar:

```
application_dcl ::= application identifier
                    ("{"applicationbody_dcl"}")
applicationbody_dcl ::= (contains_dialog ";")*
contains_dialog     ::= contains scoped_name
```

The `identifier` and `scoped_name` are IDL grammar productions. The *contains* dialog represent the dependency contains described above. An example of the application stereotype described in the CML language is shown next:

```
application appExample
{
   contains dialogExample;
   contains dialogExample2;

}
```

This example shows an application that contains two dialogs as CML representation

6.7.5.1 Dialog CML grammar extension

The dialog may not have any attributes or methods and is described by the following EBNF grammar:

```
Dialog_dcl ::= dialog  identifier ("{" "}")
```

The dialog has an empty body. An example of a dialog described in CML is given here:

```
dialog dialogExample1
{

}
```

6.8 Model layout class- generator (emitter)

The CML file that describes the UML model know contains both the system architecture as well as the view of the client with the user interfaces that it contains. Both these views are used by Multidev to generate the client. The two different views are presented in two different modules in the UML model. The process of separating them when Multidev reads them as input is done by a simple naming convention: The definitions of the dialogs that are used by the client should reside in a module with the name *system*. The COMDEF module does not have any restriction on the name used.

The code generation tools developed with COMDEF are used to generate java classes, which instances represent the two different views of the system, the COMDEF view and the Multidev view. This is done by an emitter that parses the CML file and generates two java classes. Figure 6.6 illustrates this process.

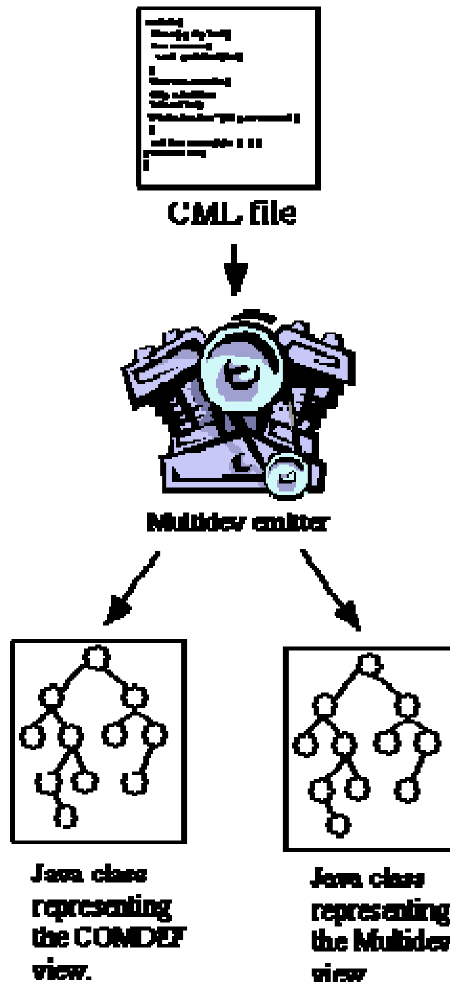

Figure 6.6 – The emitter process

The java classes hold all the information about the different model information of both the COMDEF view and the Multidev view. This is done by using a java package with classes that describe the different modeling elements used in COMDEF and Multidev. The models are stored in a tree structure. (In future versions the output of the emitter should be XML documents, but java was chosen because the COMDEF emitter already emitted Java and hence there was no need to develop an interpreter in the Multidev modeling tool)

6.9 Summary

This chapter has introduced Multidev. The development process that should be used with Multidev and the connection to COMDEF was described. The extensions to COMDEF are described in detail to give the reader an insight in how Multidev is able to use the tools that exist in the COMDEF framework to retrieve the information needed from the UML model.

Chapter 7

Multidev models.

This chapter describes the models that are used in Multidev to describe the user interface and application logic. The different models and the relationship between them are described.

Multidev uses three models to describe the functionality and looks of the user interface. These models are:

- Domain model
- Task model
- Presentation model

7.1 Domain model

The domain model describes the structure of the underlying application. This structure is described by the different objects, with their attributes, methods and relationships, that combined constitute the system. In addition, the domain model also includes auxiliary data types that may be required to describe transient data. Examples of such data may be data types needed to manipulate the data input via the user interface before it is used with the underlying system.

In Multidev, the domain model is the COMDEF system model. This model describes the services, entities and events, with their attributes, methods and relationships. For the client application the underlying systems is realized through the user services, which are also included in the domain model, and serves as the entry point to the distributed information system. The domain model also serves as the entry point of modelling in Multidev, which means that there must exist a domain model in Multidev before any modelling activity can begin.

The domain model is visualised as a three structure in the Multidev modelling tool. Figure 7.1 shows the domain model as it is presented in this tool.

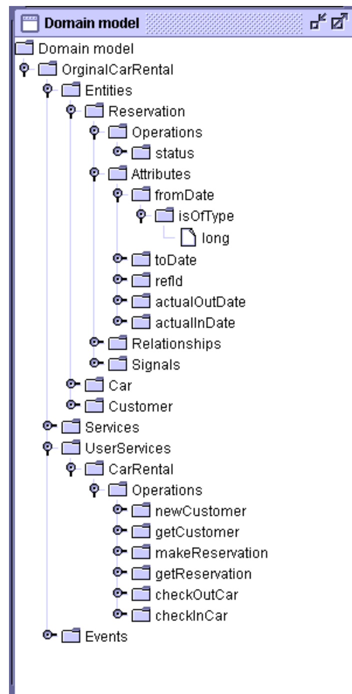

Figure 7.1 – Screen shot of the domain model from Multidev modelling tool.

As the figure shows the different objects are represented as nodes in the three. A class has child nodes that hold the different attributes, methods and relationships of the specific class. Internally in the Multidev tool the different classes in the domain model tree are represented as objects that are used to generate state objects to be used in the task model. (The task model is described next). This is done by a dragging a node and dropping it on the selected task.

7.2 Task model

The task model describes the different task that the client application must be able to accomplish. The task model has relations to both the domain model, and the presentation model. There are different ways to describe task models, and different definitions of tasks. The following definitions are used to define the tasks in the task model used in Multidev:

In [Storrs, 1995] a task is defined as follows:

A task is a goal together with the ordered set of tasks and actions that would satisfy it in the appropriate context.

In the same paper a goal is defined as:

A goal is an intention to change or maintain the state of an artifact

Storrs uses these two definitions to derive what is needed to describe a task.

- One goal
- A non-empty set of actions or other tasks which are necessary to achieve the goal
- A plan of how to select actions and task
- A model of an artifact, which is influenced by the task.

The task model proposed in Multidev is based on the task model used by Teallach [Griffiths et al., 1999], and has the same structure as other MBUIDE's (Model Based User Interface Development environment) such as Adept [Markopoulos et al., 1992], Mastermind [Browne et al., 1992] and TADEUS [Schlungbaum, 1998]. The task model is described as a hieratic structure of tasks. There are two different types of tasks, primitive tasks and composite tasks.

According to the definition of a task presented above, a task has subtasks and actions. In the task model, a composite task includes a set of subtasks and/or actions. An action is a primitive task and is divided into interactions and actions. An interaction is representing some kind of human computer interaction between the end-user and the user interface, while an action is something that is performed by either the system or the end user.

As described in chapter 5, it's advised that the methodology used when developing a system with COMDEF should be use case driven. These use cases will identify the different tasks that the application shall provide user interfaces for and the user services must provide the functionality from services that makes them able to accomplish these tasks. A particular use case may be described as a high level task [Pinheiro da Silva, Paton 2000 #1] and these will be reflected in the task model as composite tasks and primitive tasks. In the task model tree structure, a primitive task is a leaf node of a task.

As mentioned, a composite task is made up of other composite tasks and/or primitive tasks. The subtasks of a composite task must all reach their goal in order for the super task to reach its goal. A supertask can require that its sub task acquire their goal in a certain order. In order to specify a certain behavior, the composite tasks are categorized in the same fashion as the Teallach [Pinheiro da Silva et al., 2000] tool. The sub tasks of a supertask can be carried out in on of the following ways:

- Sequential: The subtasks must be performed in the specified order (top down in the task model tree), and all subtasks must reach its goal for the supertask's goal to be achieved.
- Order independent: The order of witch the subtasks are executed is not of importance, but all sub tasks must be completed for the supertask's goal to be achieved.
- Repeatable: The sub tasks must be repeated a specified number of times or until a condition is satisfied.
- Concurrent: The sub tasks must be performed in parallel. All sub tasks must achieve their goals before the goal of the super task is considered achieved. There are two kinds of concurrent tasks. Truly concurrent tasks occur at the same time, while interleaved task have only one sub task at a particular time, but all are progressing.
- Choice: The user must decide which of the subtasks that is to be performed. For the super task to reach its goal, the chosen sub task(s) must be completed.
- Optional: Zero or all of the subtasks must be completed in order for the supertask to achieve its goal.
- Conditional: A choice that is depended on a condition must be made between subtasks. The chosen subtask(s) must be completed before the supertask has achieved its goal.

A state object refers either to an instance of the domain model or to a presentation model class. (The presentation model is presented in the next section.) When a state object is created, it is given a uniquely name. After the state object is created and associated with, either a domain- or a presentation class, its operations can be invoked. The state objects are used to link the task and the domain model. When one or more state objects are associated

with a task, then these objects can be used to link the task model and the domain model together.

As an example, consider the task of retrieving the price of a particular stock. The task is depicted in figure 6.7. The name of the task name is "*Retrieve stock value*". The task is defined as a sequential task and a state object with the name "Retrieve.stock.value.*stockList*" is associated with the task. This state object is referenced to the domain class called stockList as the figure shows, which has an operation with the following signature:

```
String getValue(String ticker)
```

The goal of the task is to retrieve and display a value of a particular stock identified by a ticker and the goal is reached by accomplishing the sub tasks in an sequential manner.
A sub task called "*specify ticker information*" is added to the super task. The goal of this task is to get the ticker information from the user. This task is an interaction task where the user will give the ticker as an input. The output of this sub task is associated with the input parameter of the *getValue(…)* operation through the state object.
The next sub task is an action task named "*Get stock value*". The goal of this task is to invoke the *getValue(…)* operation on the *stockList* object through the state object. The last subtask is an interaction task whose goal is to display the stock value. This sub task is associated with the return value of the *getValue(…)* operation through the state object.

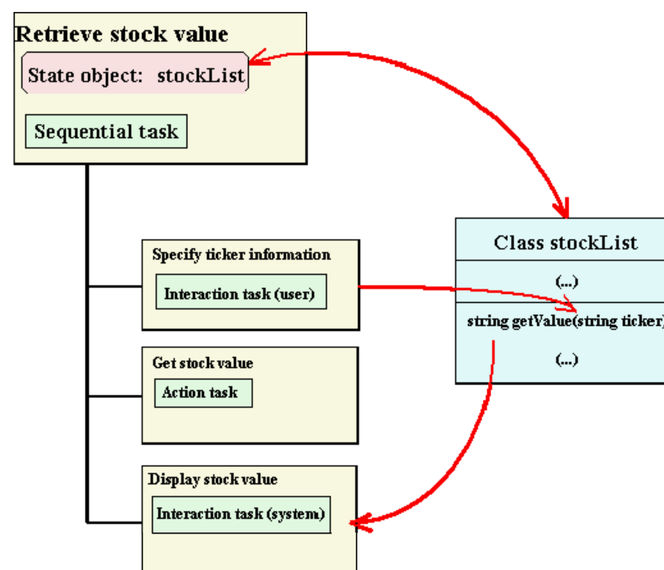

Figure 7.1 – The task model for retrieving a stock value.

As described, the *Retrieve stock value* task has to let all its sub tasks reach their goals before it has reached its goal. The state object is in this case associated with the main task, and the sub tasks are used to invoke and retrieve the data from one of its methods.

When the task model is at a stable state, the next job is to model the presentation model, which links the different tasks to presentation units, namely dialogs. The presentation model is described next.

7.3 The presentation model

When there exist a task model, the modeling of the dialog layout can begin in the presentation model. The dialog model is at the beginning of dialog modeling only a

skeleton model, and the different task must be mapped to the appropriate dialogs. The different modeling items that can be used in the dialog modeling are presented as a UML metamodel. In other words, the presentation model is an instance of the dialog metamodel. Not all tasks require dialogs as they might have nothing to retrieve from or display to the end user. The dialog metamodel is described in the next sections.

7.3.1 Dialog meta-model

The dialog metamodel is a UML profile for modeling user interfaces. The dialog metamodel is based on the dialog objects defined by Genera in the tool Genova 6.1, described in chapter 4. The metamodel in Multidev visualize the different dialog objects as stereotypes and express the relationship between them. The metamodel is used implicitly in the graphical tool provided with Multidev. The dialog-modeling tool in Multidev is not a UML modeling tool. The reason for this not expressing the dialog model in UML is that the dialog model is partially generated at modeling time by the object selection done by the developer. This would be difficult to do in a commercial UML modeling tool. Instead, a tree view of the dialog model was chosen. This will be elaborated more in section Chapter 8. The conceptual Multidev dialog metamodel is depicted in the figure 6.7

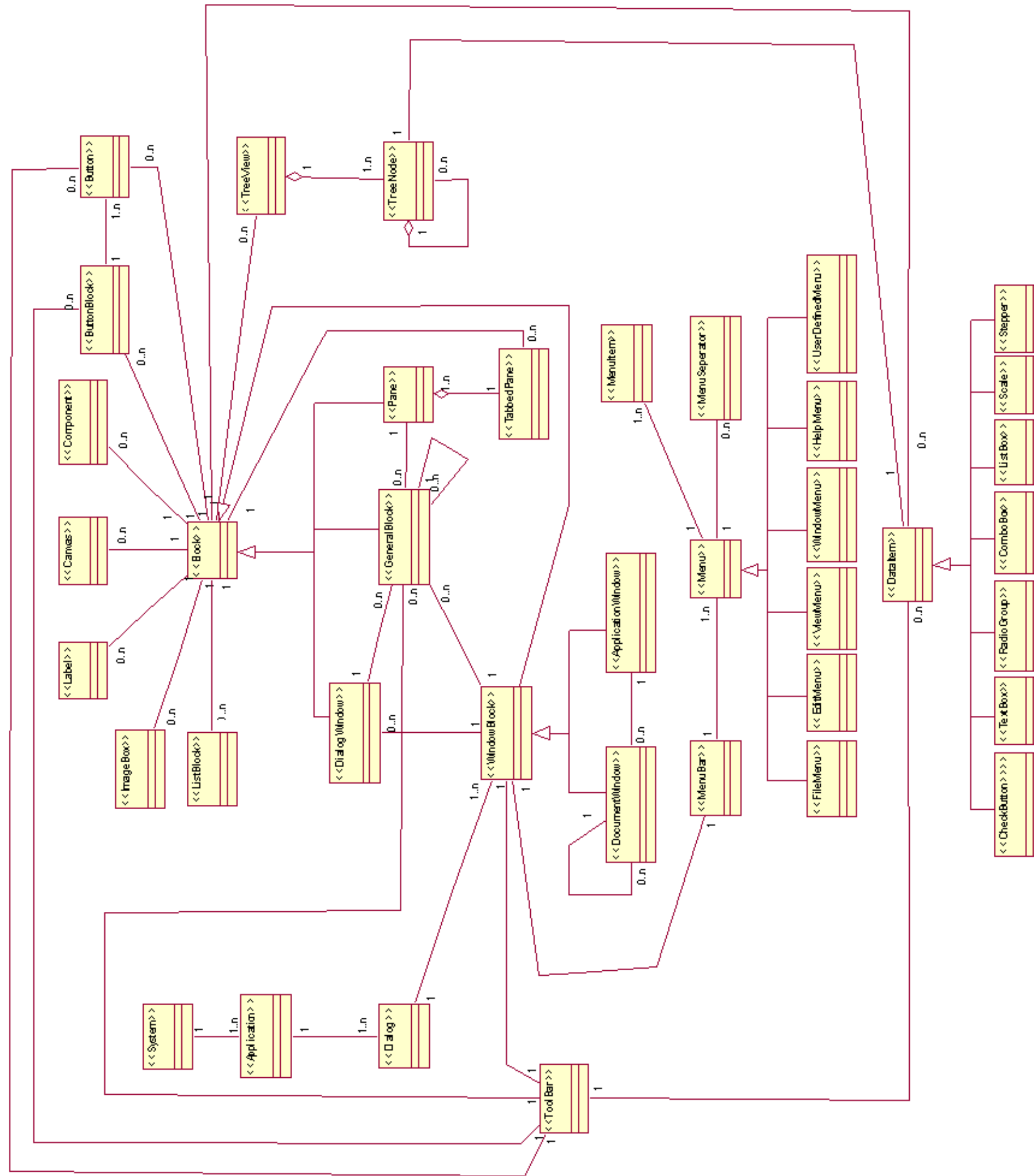

Figure 7.2 – The Multidev Dialog Metamodel

The stereotypes in the metamodel makes it possible to model user interfaces for graphical window based user interfaces. The various elements could be used to model non graphical user interfaces as well, as long as the elements are mapped and given meaning on the specific platform. The main reasons for having a metamodel is to be able to describe the different parts of the dialog model precisely in order to code generate the user interface. The metamodel seek to be transparent to what kind of platform the user interface is intended to run on, with the exception, that the platform must support graphical user interfaces. The modeling environment in Multidev is as mentioned not an UML environment, but since it uses a UML metamodel implicitly, it is possible to map the dialog model to a UML model if desired.
The main parts of the metamodel are described in the next sections.

7.3.2 Dialog metamodel main concepts

In Multidev, a dialog structure is described as a *system* that contains one or more applications. An *application* is a collection of *dialogs*, which again contain different user interface objects needed to fulfill the task of that particular dialog. The structured information of which dialogs an application contains is modeled in the UML modeling tool and kept together with the COMDEF model, as described in section 6.3. This model is the base for modeling in the modeling tool in Multidev and will be elaborated in section Chapter 8. In the next sections the different dialogs components in the metamodel will be described in different views of the metamodel. These views are incomplete views, which only show certain parts of the metamodel. Some associations are left out of the figure to simplify the view.

7.3.2.1 Dialog

Figure 6.8 shows the *dialog component* and its associations. The structure of the user interface layout is as mentioned structured as a *system* that contains one or more *applications*. An *application* consists of on or more *dialogs* which in turn is made up of one *application window* and one or more *document windows*. Application window and document window will be described in section 7.3.2.3.

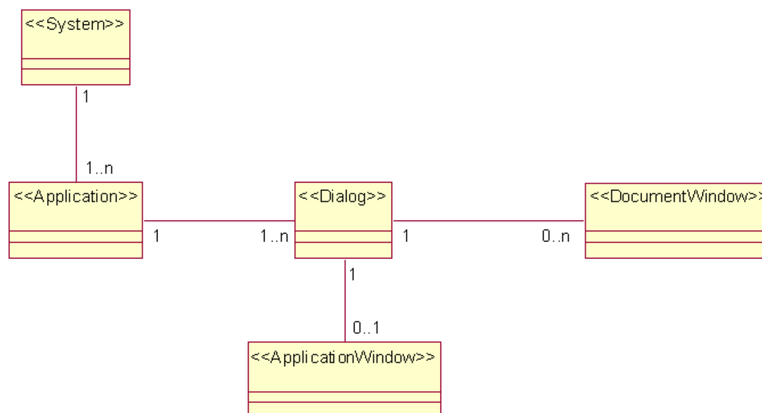

Figure 7.3 – The dialog object and its associations.

7.3.2.2 General block

The general block is one of five blocks in the Multidev metamodel. Figure 6.9 shows all objects that are derived from the block element.

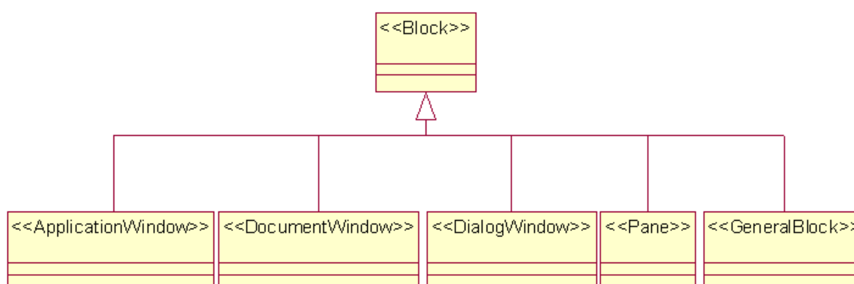

Figure 7.4 – The block object.

A block is a component capable of containing other user interface objects within it. Some of these blocks may also contain other blocks within them selves. Figure 6.10 shows the general block and its associations.

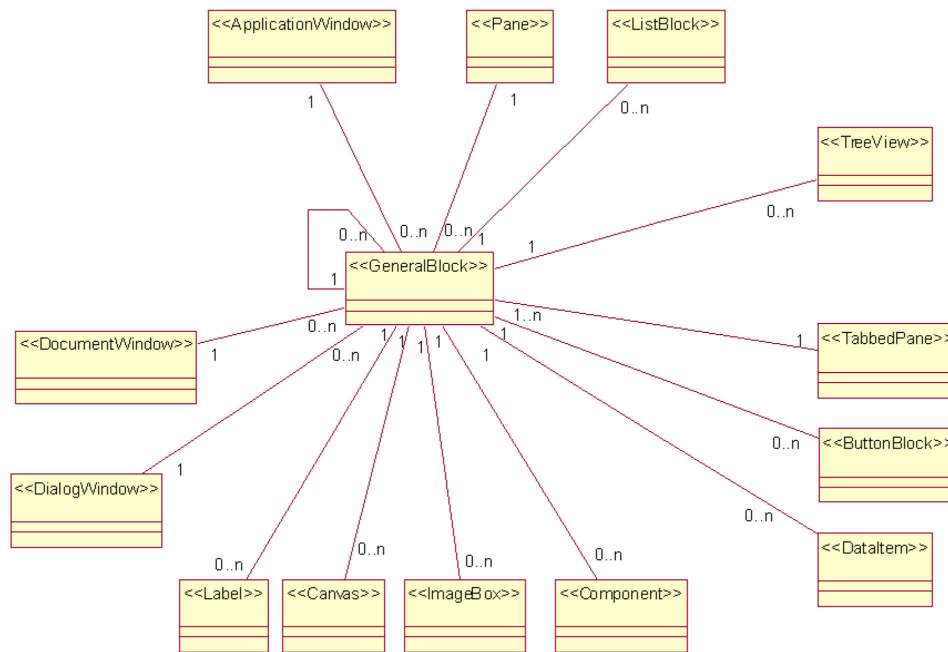

Figure 7.5 – General block.

The general block is used to group together user interface objects. This is done when the dialog objects are logically related or if there are some other reason for grouping the objects together. As the figure shows, the general block may be contained within an application block, a document window, a dialog window or a within another general block. All the five blocks in Multidev are derived from the block component, which can contain the following objects:

- **Tree view**
  This object represent a tree. Trees consist of one or more nodes. The tree view object will be described in chapter 7.3.2.8.
- **Tabbed pane**
  A tabbed pane is an object that is used to structure the user interface into different panes. The object is described in section 7.3.2.7.
- **Button block**
  A button block is a block that only can contain buttons. The button block will be described in section 7.3.2.5
- **Data Item**
  A data item represents an item that display some kind of data, such as a text box. Data item is discussed in section 7.3.2.4.
- **Component**
  A component represents a third party user interface component.
- **Image box**
  An image box is a container for images.
- **Canvas**
  A canvas is used to set a side some space in the user interface, where the client application may paint. The canvas is used for advanced graphics.
- **Label**
  This object represents a typical label.
- **Dialog Window**
  A dialog window is also a container and represents a dialog box used to get feedback or to inform the user.

- **Button**
  A standard button that is used by the end user to invoke an action from the application

7.3.2.3 Window block

Two stereotypes derive from the window block. These are document window and Application window. An application window has the purpose of being the main window of the client and a dialog structure must have one such window. Figure 6.10 depicts a view of the metamodel that shows the window block and its sub classes. A window block is also a generalization of the block component.

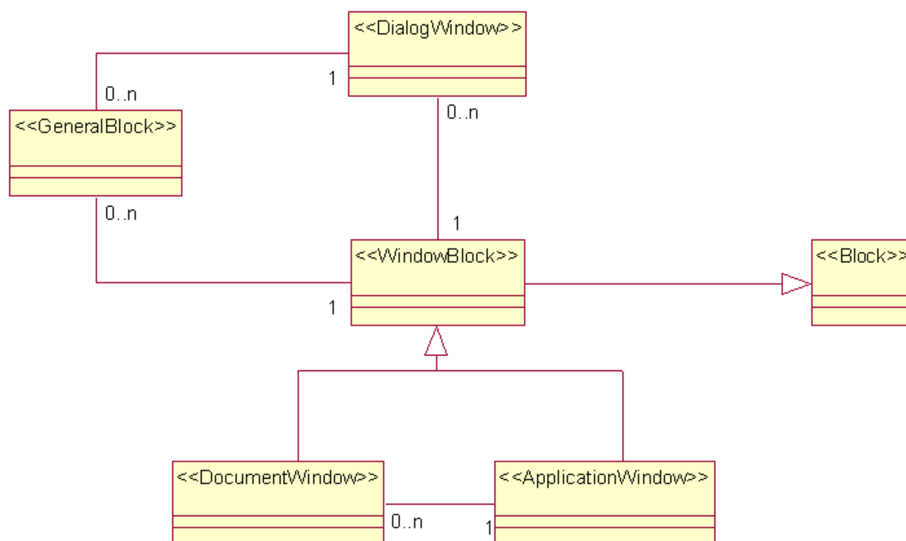

Figure 7.6 – Window block components.

An application Window may contain other document windows within it. These are activated from the application window. Both the application window and the document window may contain a toolbar and a menu bar. This will be discussed more in section 7.3.2.10

7.3.2.4 Data items

A data item can be bounded or free. A bounded data item is used to represent data, such as values from attributes of classes while a free data item displays values that are not persistent. Free data items can be used show results of computations from other data items in the user interface. Figure 6.12 depicts the data item and all object that are derived from it. As the figure shows, a data item may be contained in window blocks, toolbars, panes, general blocks, dialog windows and tree nodes. Bounding a data item is done at modeling time. If a data item is not bounded it is regarded as a free data item.

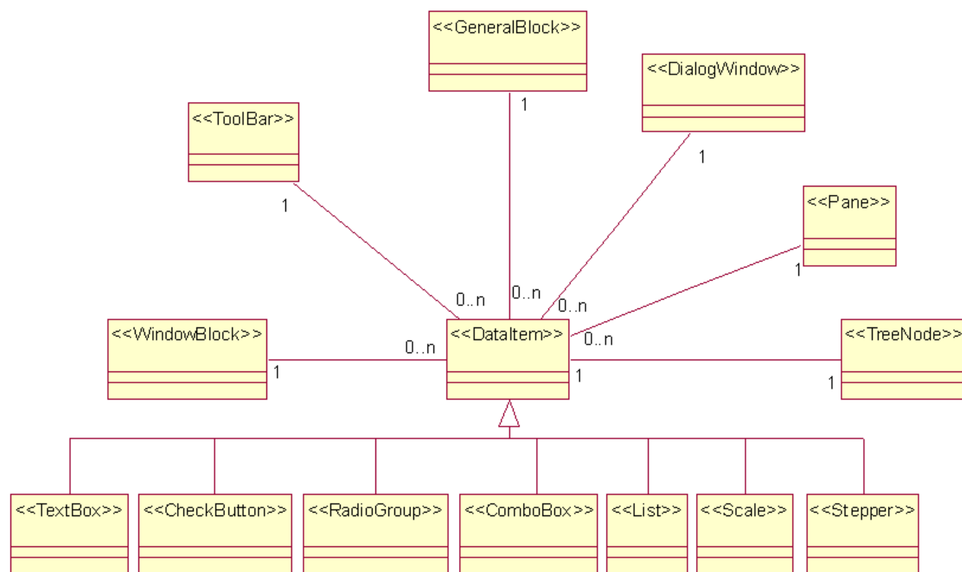

Figure 7.8 – Data Item.

7.3.2.5 Button block

A button block is used to group together buttons. Figure 6.13 depicts a model view of the button block. A button block may be contained in an application window a document window, a toolbar and in a general block. There must be at least one button in a button block.

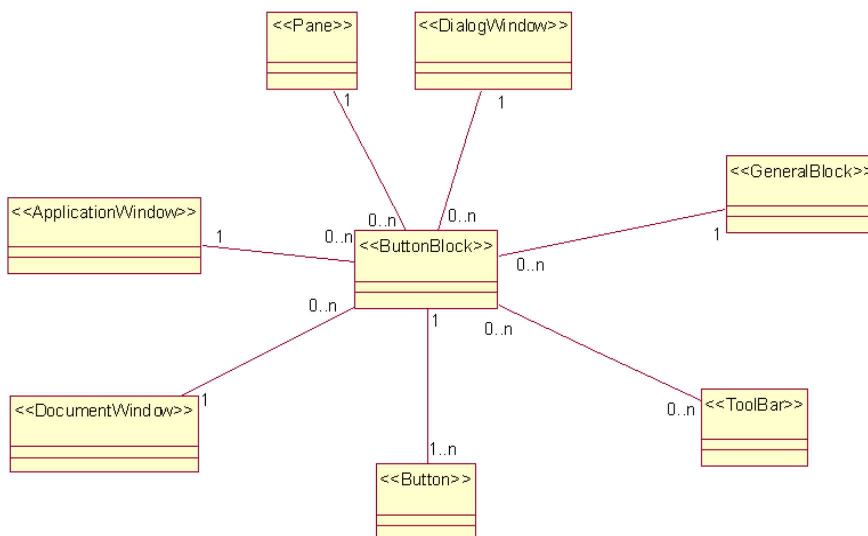

Figure 7.9 – Button block.

7.3.2.6 List block

A list block is used to display several instances of the same set of data in a multi-column list box. Figure 6.14 depicts the list block object. It can be contained in a pane, a general block, a dialog window, an application window or a document window.

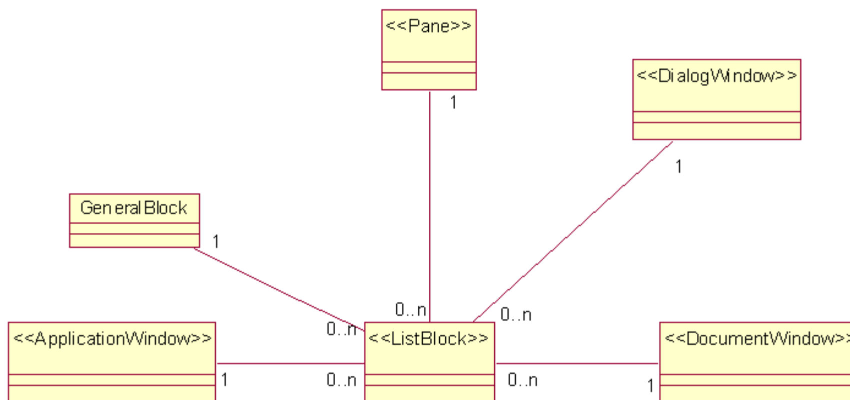

Figure 7.10 – List block.

7.3.2.7 Tabbed pane

A tabbed pane represents a way of structuring a user interface. Figure 6.15 depicts the tabbed pane and its associations. As the figure shows, a tabbed pane is made up of one or more panes, and can be contained by application windows, document windows, dialog windows and in general blocks. A pane is a block object and in the tabbed pane the different panes can be selected by clicking on the title of each pane.

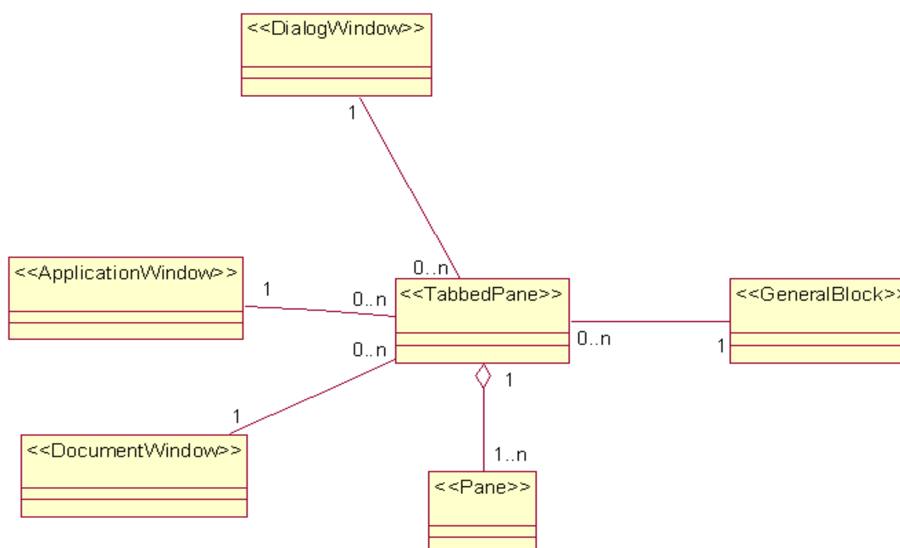

Figure 7.11 – The tabbed pane component.

7.3.2.8 Tree view

The tree view object is a standard user interface tree. It must contain one or more tree nodes. A tree node may only contain other tree nodes and one data item. Figure 6.16 depicts the tree view object. The tree view object may be contained in a general block, a pane, a dialog window, a document window and an application window.

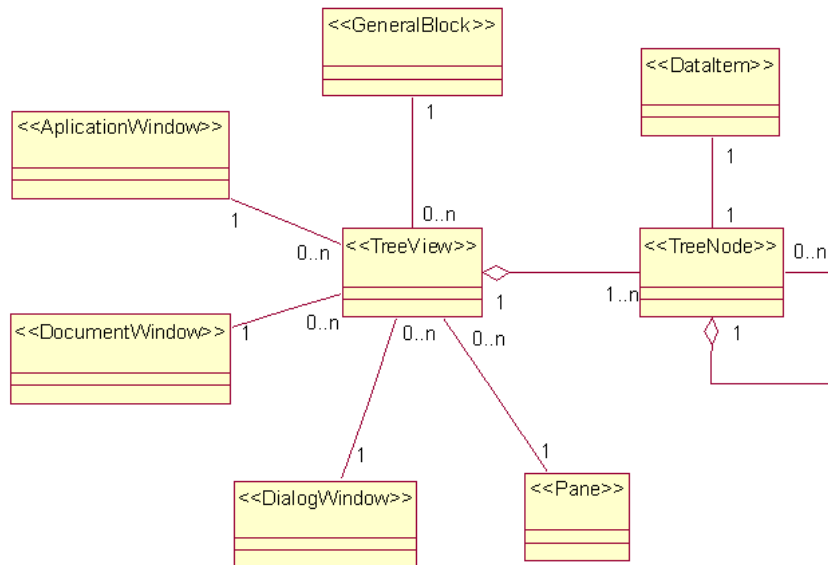

Figure 7.12 – The tree view object.

7.3.2.9 Menu bar

Both application window and document window may contain one menu bar. A menu bar is made up of one or more menus. The most common menus are defined, and a user-defined menu is available. The menus consist of user defined menu items and menu separators. Figure 6.17 shows the menu bar object and its associations and the different menus that are defined by the Multidev dialog metamodel.

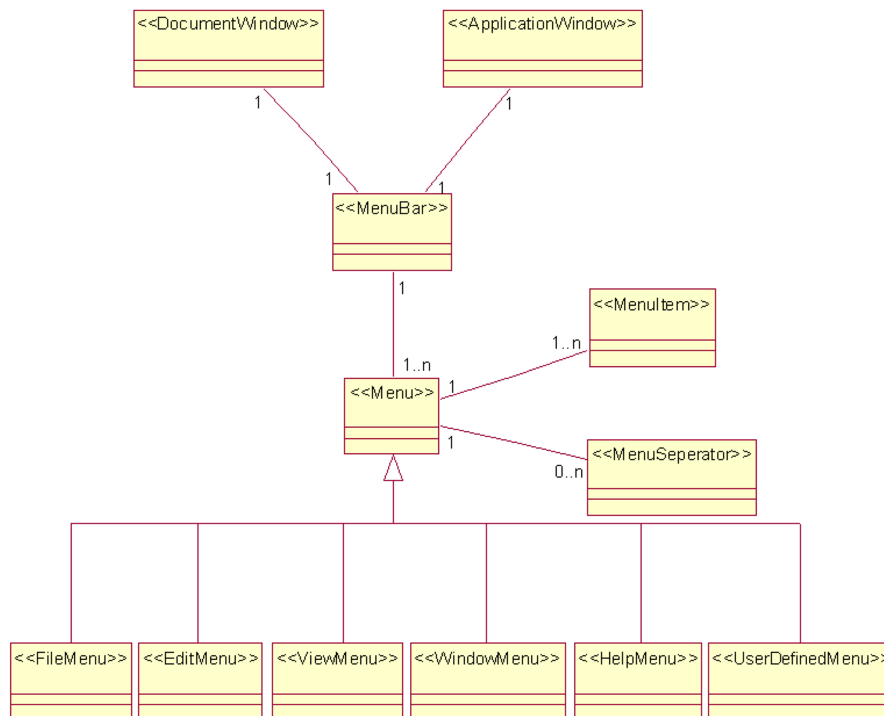

Figure 7.13 – The menu components

7.3.2.10 Toolbar

As with menu bar, a tool bar is available to both application window and document window. Figure 6.18 depicts the toolbar object. A toolbar is made up of buttons and data items. These can be grouped by using general blocks and button blocks.

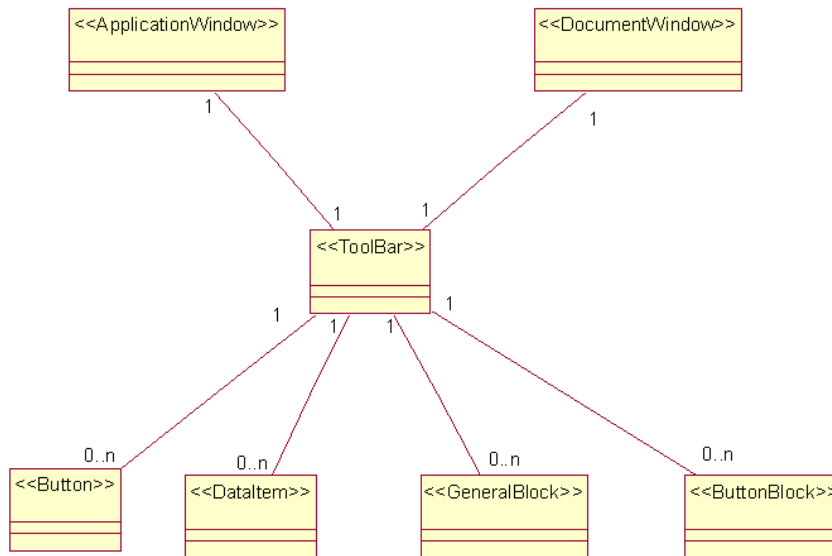

Figure 7.14 – The toolbar.

7.3.3 Dialog object properties

All the different objects of the metamodel have a set of properties that describe their appearance and their connection to data sources. The different objects have different properties depending on what purpose they serve. Not all properties are valid on all platforms. The different properties of the dialog objects are not valid on all platforms that can be supported. A stand alone application running on java or C++ will be able to display all the properties of the dialog objects of the metamodel, where as a web application do not have tabbed pane as a standard user interface object provided by a toolkit. Multidev describes how the different objects are to be presented on the supported platforms. The metamodel is used to model an abstract model of the user interface. Multidev maps the abstract model to a concrete instance at code generation time. This is done by describing how to map the different object to a selected platform, and by describing how specific layout pattern will look in a particular environment.

7.3.4 Mapping the tasks to presentation

When the task model is at a stable state it is mapped to the presentation model, using the different dialog objects defined in the dialog metamodel. The example in section 7.2 showed a task model representing the task of retrieving a specific value of a stock by specifying the ticker of the stock. Figure 8.4 show the corresponding presentation model to this task model using the Multidev tool.

7.4 Relationships between models

All the different models are used to be able to code generate the user interface. The domain models represent the underlying distributed system. However, the domain model does not include information about the information flow of the application. This information is modeled in the task model. The link between the domain model and the task model is the state object. A state object includes one object from the domain model. The task model describes the different tasks which goal is to change the state of the various state objects. That is, change the state of the domain object that is wrapped in the state object. The

different tasks are represented as dialogs in the presentation model. This model is built using the dialog metamodel.

7.5 Summary

This chapter has introduced the three models that are used in Multidev. These models are the domain model, the task model and the presentation model. The domain model represents the underlying distributed system that is realized by the user service. The task model describes the different tasks the client application must provide. The presentation model represents the user interface layout. The presentation model is an instance of a dialog metamodel that contain different dialog objects. The relationship between the models was also described in this chapter.

Chapter 8

Dialog modeling and code generation

This chapter describes how dialog modeling is done in Multidev. The tool that is developed as part of the framework is presented and how the code generation is done is presented.

8.1 Multidev modeling tool

Multidev provides a graphical modeling tool to aid the developer in designing the structure and content of the different models described in chapter 7. The modeling task can begin when there exists both a domain model (the COMDEF model) and a skeleton of the user interfaces. As explained in section 6.8, the Multidev emitter will generate two java classes containing the domain- and presentation skeleton model, needed by the Multidev modeling tool. When the tool starts up it check that these two java classes exist. If the classes are not compiled Multidev will do this before using them.

8.1.1 Modeling environment

The modeling tool is developed in java and is a graphical tool where the modeling is drag and drop based. Figure 6.19 shows a screen shot of the modeling tool before any modeling as been done. The toolbar on the left, display the different tasks that can be used to build the task model. The toolbar on the top display the dialog building blocks that can be used to build the presentation model.

The three windows each contain a tree structure, which represents the models described in chapter 7. At startup the domain model is the only complete model and is as mentioned the entry point of modeling in Multidev. The left tree structure represents the system model or the CODMEF view of the System. The middle tree structure represents the task model, and is empty at start up. The right tree structure represents the presentation model that was defined in the same model as the COMDEF system model. At start up this model is only a skeleton model of the different dialogs. The menu contains some of the usual menus used in windows environments as well as the Multidev specific menus.

The menu lets the user select which layout pattern to use. Multidev provides to different layout patterns for the dialog layout. This choice will affect the way Multidev generate the proposal of a dialog layout. A window layout will group the different dialogs in the right tree of figure 6.19 as windows. The other layout pattern is a tabbed pane layout, which will group the different dialogs as panes. The proposal may be over ridden during modeling time.

A special menu brings up a window for selecting which platforms to generate source code. In current version of Multidev, it is only possible to generate java. If java is selected then Multidev produces a stand-alone application that has an internal user interface renderer for rendering and displaying the interface. The renderer handles the events that are triggered by the user when he pushes a button or in other ways interact with the user interface.

The nodes of both the task model and the presentation model have a set of properties that can be set by double clicking on the particular node. For the task models, these properties consist of selecting a variable or a method of a state object that should be associated with a sub task. For the presentation model, the different properties are used to modify the

appearance of the user interface objects. At the present time, there is no consistory check of the different models, making the responsibility of the user to make sure that the models are sound.

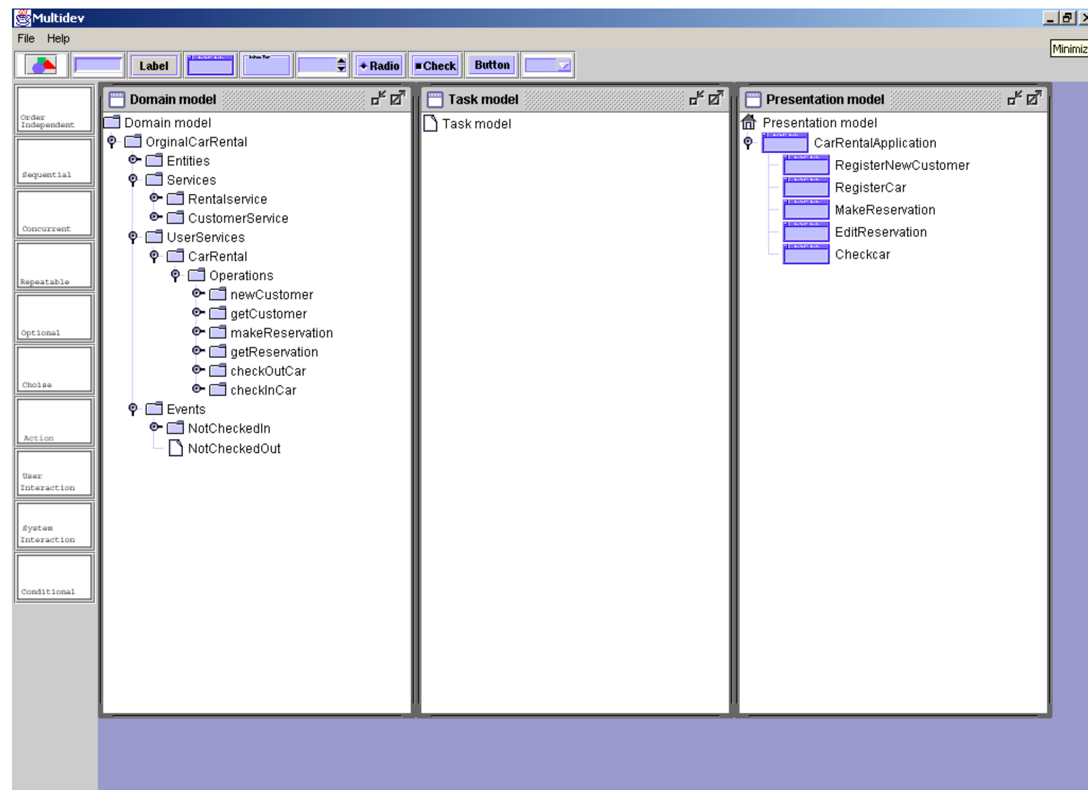

Figure 8.1 – A screen shot of the user interface tool

8.1.2 Modeling concept

The user service is the link between the client application and the underlying distributed system. All the systems use cases must be accomplished by calling the methods described in the user service or user services. Therefore, the use cases are central in the modeling of the system. All use cases of the system have to be implemented using one or more of the methods offered by the user services. In fact, the user services are really the product of the use cases, as these will have unveiled what services that are needed in the early stages of system development. Through the use cases, the different services and entities are identified and the userservices provide a collection of these services to the client application. A user service might be viewed as a top-level use case and the operations it provides represent the functionality of finer granularity use cases on their own. In addition, the combination of user services may describe a use case. In Multidev, the modeling is based around the different operations of the user service or user services. The user service is in most cases responsible for delivering the state object to a task and/or to pass it down from the task to the underlying distributed system. Sometimes, however, the state object may be an object that is only used in the user interface and not on connection to the user service. In these cases the use case are not reflected by the user service.

8.2 Code generation

The goal of the code generation is to code generate all of the application, based on the three models presented in the previous chapter. There are two different code generators. The first handles the user interface files, while the second handle the code generation of the

application logic. At the time of writing, the only supported platform is java for the application logic, but support for other languages should be straight foreword to implement.

8.2.1 UIML mapping

When the presentation model is at a sound state, the user interface generation may begin. The user interface generation consists of mapping the presentation model into a UIML description file, which is used to display the user interface. This is done by applying a layout algorithm to the user interface three. Besides the user interface constructs the UIML file also describes the different events that occur in the user interface, such as clicking on buttons or typing on the keyboard.

8.2.2 Layout algorithm

The layout algorithm is used to group the different user interface objects. This process is done by first finding the leaf nodes of the dialog layout tree. The size can be set by the developer by setting the dialog objects size property. If the property is not set then a default property is used. Fore some dialog objects such as labels, the property is calculated by counting the characters of the label, and the font used. When all the leaf nodes of a block component are processed, the block component layout is generated based on the size of the children.

As an example, consider a login dialog containing two labels, two textboxes and two buttons as figure 8.2 depicts.

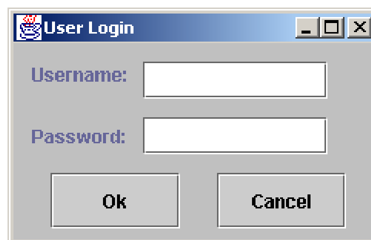

Figure 8.2 – A login dialog.

The first label is used to display the text “Username:” while the other displays “Password:” The labels are connected to a textbox in the property window of the label. The textbox representing the password is marked as a password text box in the property window. Both the labels and the textboxes are grouped in a simple block. The buttons are the standard “Ok” and “Cancel” buttons, and are grouped in a button block. Both blocks are contained in a dialog window object.

The buttons sizes are generated by calculating the size of the caption with respect to the font used. The button block is then generated by calculating the size of the buttons and adding the free space that should surround them. By default the button block has a horizontal layout of buttons, if not otherwise set in the button block property window. The size of the dialog window is the calculated by looking at the size of the two blocks and the free space that should surround them. The general block containing the labels and textboxes are placed on top and the button block is placed below because this is the order of the branches in the dialog tree. Both buttons are also bound two events that are handled by the application logic.

The corresponding presentation model will look like the one in figure 8.3.

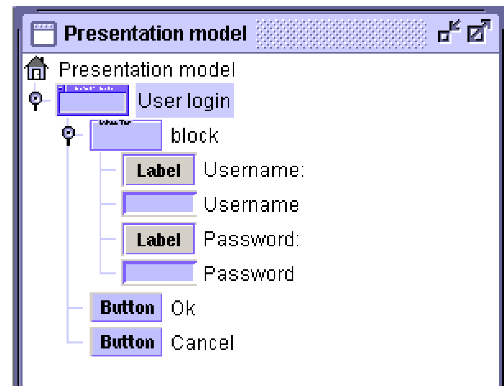

Figure 8.3 – The presentation model for the user login dialog

Each of the components in the presentation model is mapped into UIML constructs that become the user interface definition file. The mapping is done using the layout algorithm described above and with the different properties of the specific component. The UIML interface file is described with generic components matching those of the dialog metamodel. The mapping between the generic elements and the particular platform is handled in a separated code section called a peer. With proper rules, only one UIML file is needed for all supported platforms. However, in this version of Multidev there is one definition file for every supported platform.

8.3 Client Application logic

The application logic can be implemented in any supported programming language. Java is the only supported language in this version of Multidev, but porting to other languages should be uncomplicated (but may be time consuming).

The application logic has to deal with three major areas. The first is the connection to the underlying distributed system. This is code generated based on the choice of architecture. In this version of Multidev, the only technology that has been used is CORBA. The objects that are delivered to the client are handled internally in the code to produce non-CORBA objects to the client. (This should be done by the user service itself, but the current version of COMDEF does not support it).

An important part of the application logic is the generation of the code that will represent the different task of the task model. This is done a similar manner as in the Teallach tool [Pinheiro da Silva et al. 2000]. It is based on the MVC (Model View Controller) pattern [Gamma et al., 1994], [Krasner, Pope, 1988]. This pattern specifies how user interface software should be separated into components, each with a specific function. The MVC pattern defines three generic components for user interface implementation. The model component represents the domain specific software simulation or implementation of the application's central structure. The view component is responsible for anything graphical. They request data from the model and display it. A view may be super view, or a sub view. The super view will consist of one or more sub views. The controller components contain the interface between the views, models and input devices, such as a keyboard or mouse.

In the article, [Krasner, Pope, 1988], it is stated that the MVC methodology will allow programmers to write an application by first defining the classes that will embody the special application domain-specific information. The user interface may then be designed by laying out a view for it by using instances from the pre defined user interface classes.

The MVC model is used on two levels. First, every composite task is associated with an interaction component and a grouping component. The composite task is, as described in

chapter seven, made up of one or more sub tasks. The interaction component acts as the controller for the sub task(s) of the composite task. Secondly, an interaction component associated with an action task act as a controller for a state object, which in turn acts as the model. An inter action object that is associated with an interaction task may be a controller or a view. If the interaction object represents a display object, then it acts as a view. If it is an input object it act as a controller, where as if it is an object that can both display output and collect input it acts as both a view and a controller.

As an example, consider the get stock value task model described in chapter 7. The example consists of a sequential task with three sub tasks. A presentation model that corresponds to the task model is shown in figure 8.4.

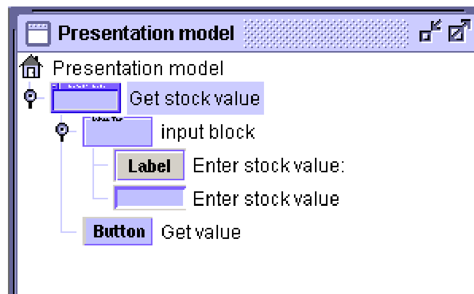

Figure 8.4 – A presentation model for the task model

A simplified view of the classes this example would generate is depicted in figure 8.5. The get_stock_value class is responsible for implementing the main method witch in turn invokes the root task to retrieve the stock information in the class named retrieve_stock_value. The retrieve_stock_value_container serves as the MVC view. The action task get value, is the MVC controller.

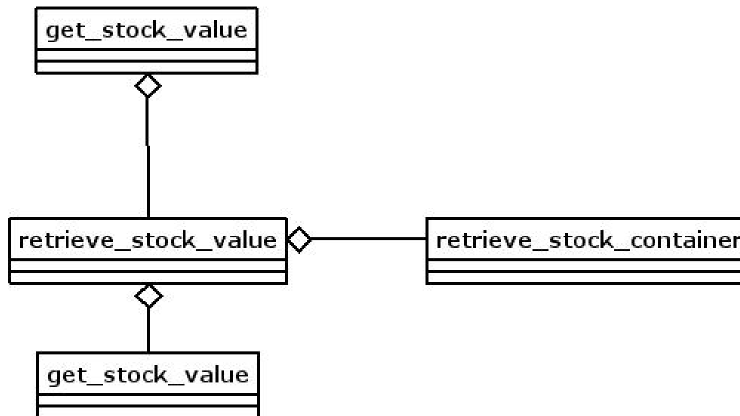

Figure 8.5 – Class overview

The container classes are responsible for displaying the user interface. In Multidev the user interface is presented through a renderer. The container classes must use the interface of the renderer to display interfaces and send and receive data and react to events.

Figure 8.6 displays the relationship between the domain, task and presentation model. Multidev allows for modeling of the task model and enhancement of the presentation model. The Domain model is set by the COMDEF system model. The initial presentation model is also imported from the COMDEF system model.

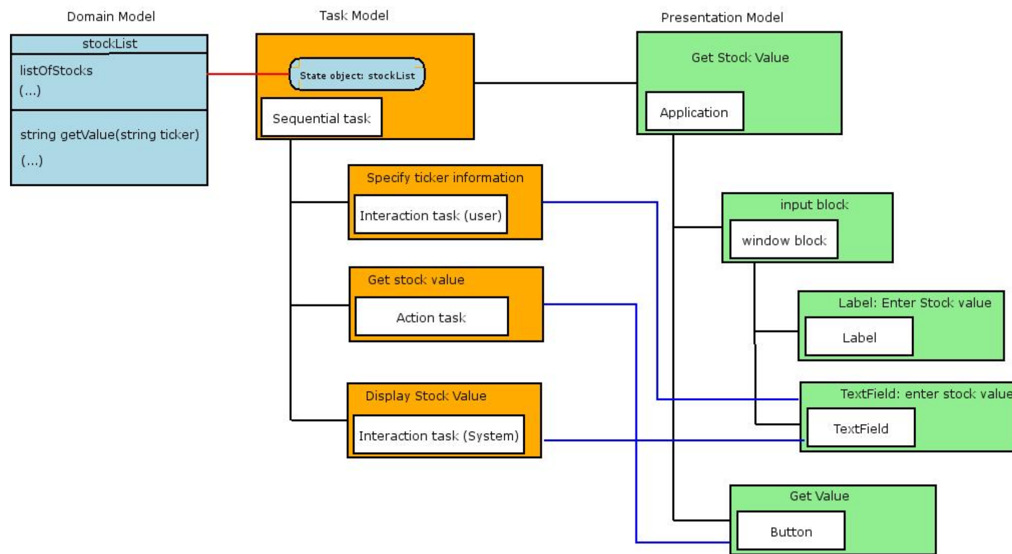

Figure 8.6 – The Domain, task and presentation model for stock value

8.3.1 Connection to the user interface

Connection to the user interface is done through a renderer. This renderer might reside inside the client or on an interface server, as described in section 6.3. The connection to the different dialog objects are done by having dummy object in client application that maps to the real dialog object of the user interface. The mapping is done in the client application by calling the appropriate method in the renderer. The renderer is responsible for reacting on the events of the user and passing them to the application where they are handled by the generated code. This differs from the Teallach approach, as Multidev adds the renderer as a layer between the task and presentation model. This is illustrated in figure 8.7.

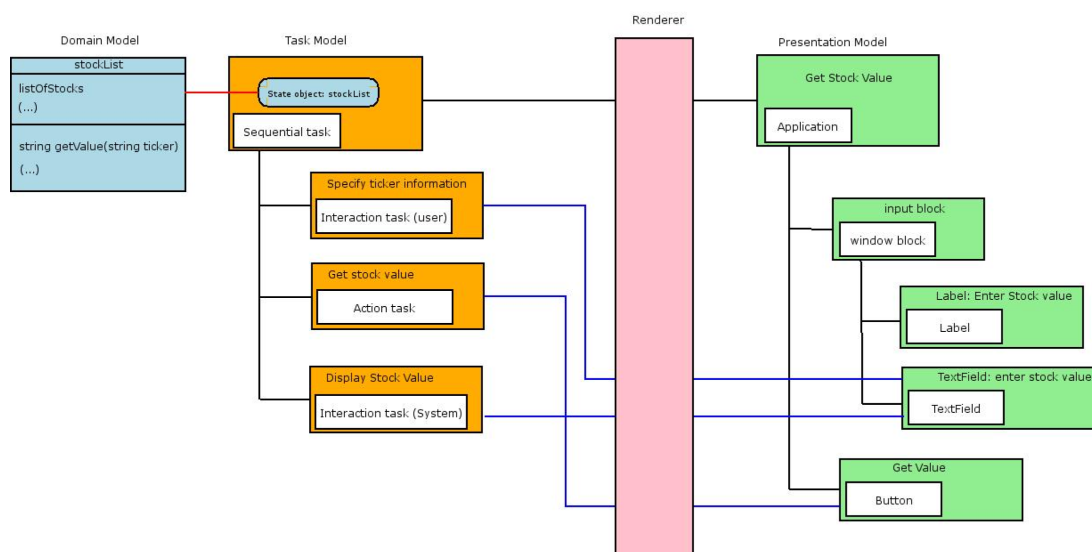

Figure 8.7 – D-T-P models with render

8.4 Summary

This chapter has described the dialog modeling and code generation from Multidev. First, the modeling environment was described. Then the building of the task model and the presentation model was elaborated. The code generation of the user interface files in UIML and the application logic was described, as well as how the events are caught and handled

Chapter 9

Case revisited

In this chapter the case form chapter 2 is revisited and the framework, described in the previous chapter, is applied to the case. The process is described and the requirements of chapter 3 will be analyzed. The case is revisited with the purpose of explaining how the Multidev framework may be applied to the case and is hence not fully implemented.

9.1 The car rental example

This example is based on a finished COMDEF UML model of a system that handles car rental to customers. The COMDEF UML model is shown below.

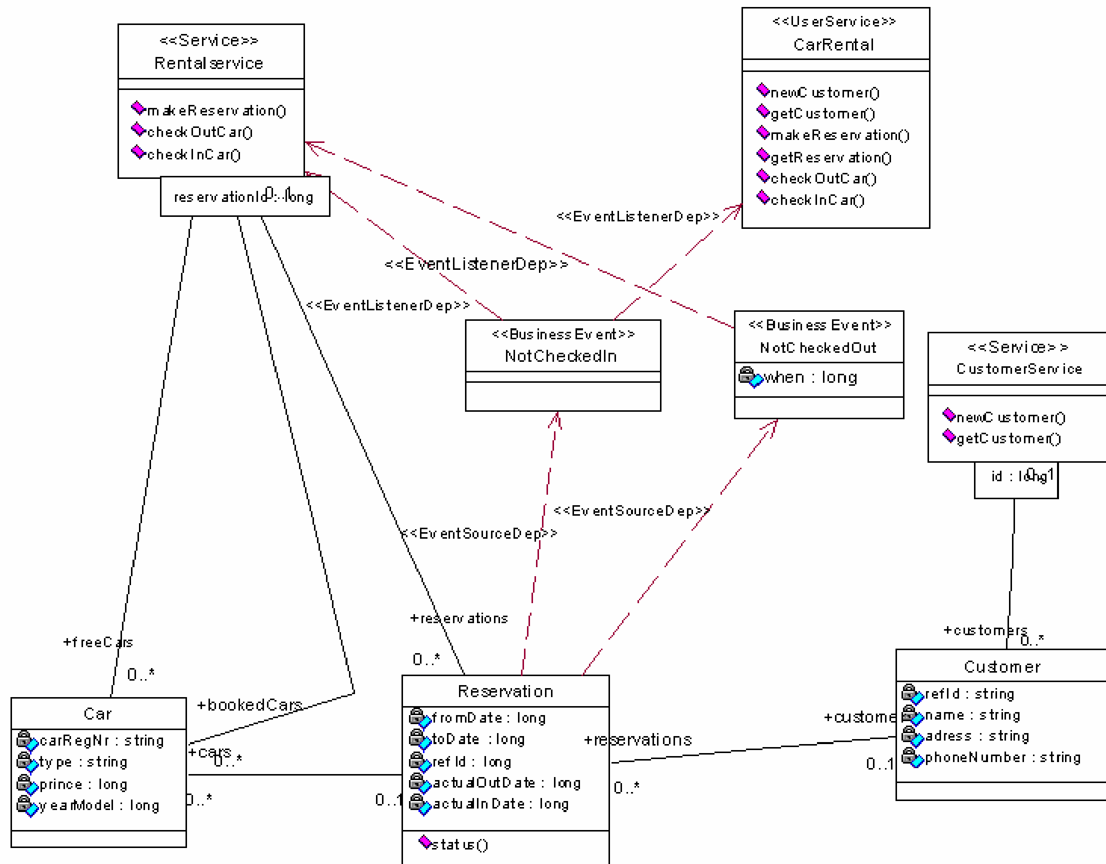

Figure 9.1 – The Car rental UML model.

The userservice typically represents the different operation that the user needs an interface to. The userservice may be seen as a collection of the operations of the services in the system. The userservice in this case consists of six different operations. These operations can be found in the customer service and the rental service. In order to get something to build a user interface from we need to do an object selection. As all ready stated the userservice is a good place so start, because the operations will typically be represent within their attributes and return types the different classes the client application must handle. The way this selection is done resembles the way it is done in Genova. First, the userservice is

found and its methods are analyzed. The operation signatures of the userservice are shown below:

newCustomer(id : string, name : string, adress : string, phoneNumber : string) : **Customer**
getCustomer(id : string) : **Customer**
makeReservation(theCustomer : Customer, theCar : Car, from : long, to : long) : **long**
getReservation(reservationId : long) : **Reservation**
checkOutCar(reservationId : long) : **void**
checkInCar(reservationId : long) : **void**

When finished selecting the objects an UML model of the client application dialogs is built based on the different UML stereotypes defined. This model gives an overview of the functionality in the different components in the system and their relationships. This model is depicted in figure 9.2 below.

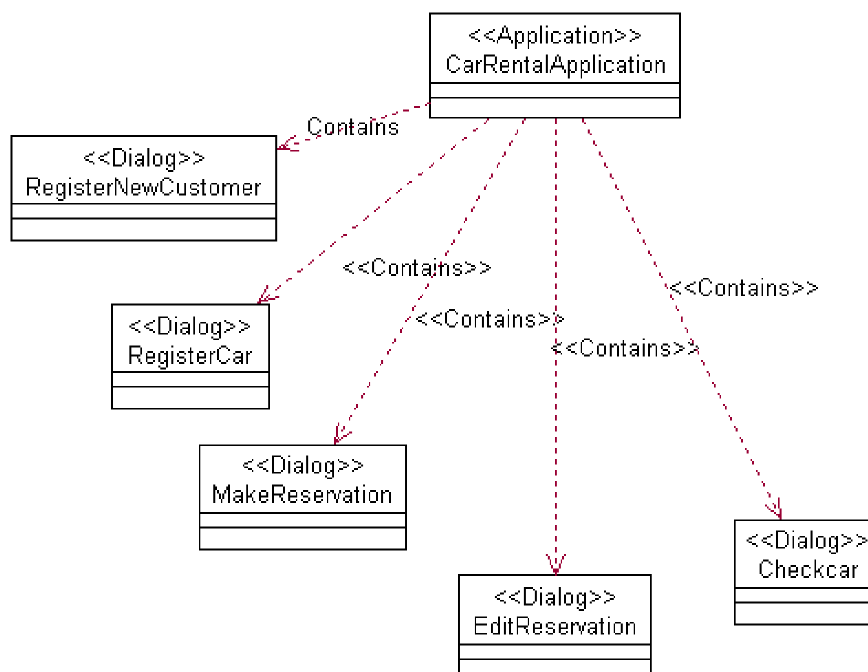

Figure 9.2 - The Dialog model

When the UML model is ready it is run through the emitter, which in turn produce to files that are used as input to Multidev. These files are:

1. Modelholder.java
2. GUIholder.java

The Modelholder.java contains the description of the userservice and the operations offered. This will serve as the domain model in the Multidev modeling tool. The GUIHolder file contains the UML model of the layout of the application's interfaces and serves as the initial presentation model.

When the Multidev tool is started these files are compiled and used to generate a three view of the domain model as well as an initial view of the presentation model.

In this case study, the systems name is “CarRental”. The system will consist of one application called “carrental” and a set of dialogs as depicted in figure 9.2. The modeling in Multidev is drag and drop based. As an example let’s consider the Register new customer dialog. The initial view of the presentation model is depicted in figure 9.3.

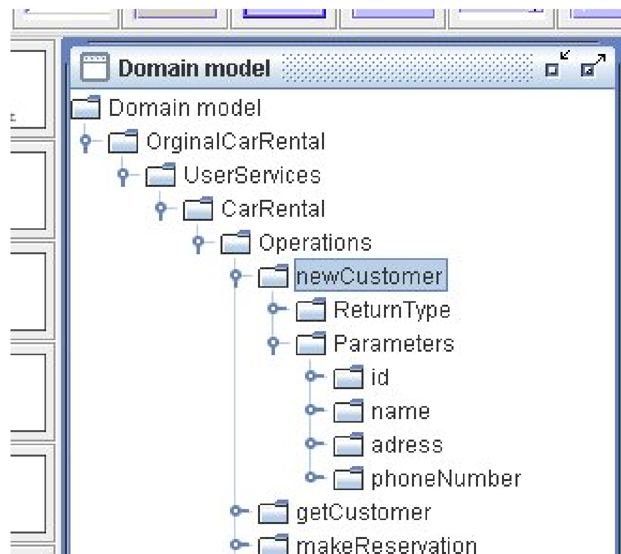

Figure 9.3 –the Domain model

As you can see the presentation model contain a dialog called new customer. To start modeling this user interface the user can drag the operation newCustomer from the domain model and drop this on the appropriate dialog in the presentation model.

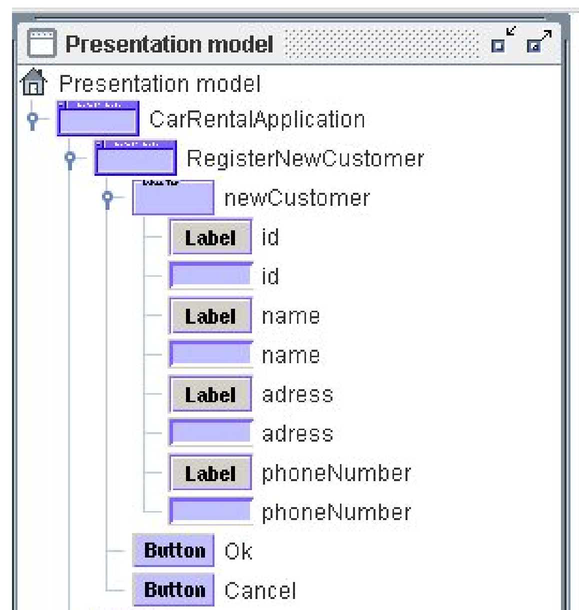

Figure 9.4 – The dialog model

Multidev will then generate a suggestion for the user interface elements that could constitute this dialog. The dialogs can be further refined by dragging elements from the top toolbar depicted inside the red circle in figure 9.5.

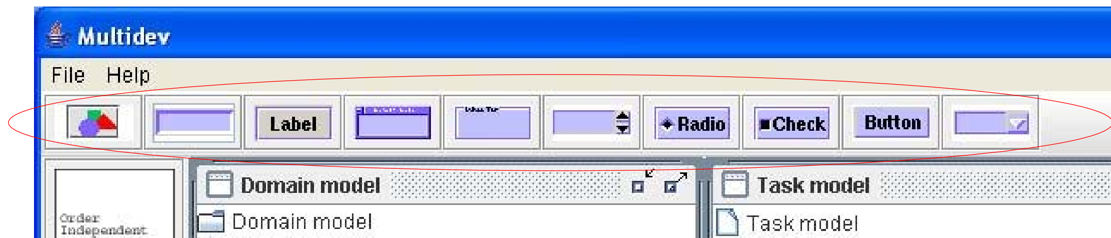

Figure 9.5 – the dialog elements toolbar.

The dialog elements in the top toolbar depicted in figure 9.5 have a set of properties that can be set by double clicking the item in the dialog model. This will bring up the Data Item definition interface where the user can select the subtype and other properties. The data item definition dialog is depicted in figure 9.6

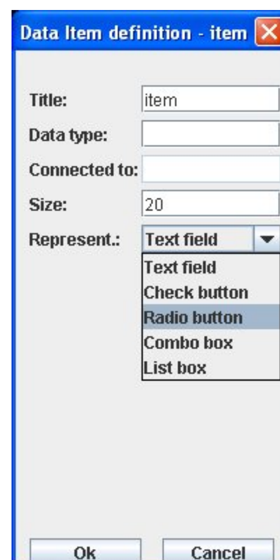

Figure 9.6 – The data item definition dialog.

The task modeling in Multidev is not complete in this version, but the concept here also utilizes drag and drop. The different task types are accessible from the left toolbar and can be dragged onto task model. This toolbar is depicted on the left side in figure 8.1. The idea is to set up a task model that corresponds to the different dialogs in the presentation model. The two models can be linked by dragging the elements from the presentation model onto the task model. Figure 9.7 displays a suggestion of the task model for the new customer dialog.

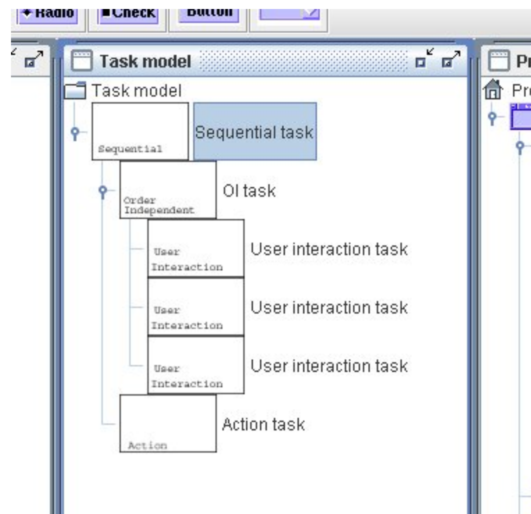

Figure 9.7 – Task model

As the figure depicts the task model for the dialog consist of a sequential task that have to tasks that must be performed in sequence. The first sub task is an order independent task with three subtasks of the type user interaction and these are to be linked to the input dialog object for getting the name, address and phone number from the user. The Action task is linked to the ok button. Instead of having a button indicating the completion of the three user interaction tasks, the system assumes that these have been completed when the ok button is pressed and use what ever value is in the linked elements in the dialog model. The cancel operation is not represented by a task, as this will only close the open dialog, and is handled by the client.

9.2 Code generation

When the task model and the presentation model are finished the code is generated by invoking the emitter. This is done through the file menu. The emitter generates an UIML file for each dialog in the presentation model as well as UIML files that map the different UI parts to client operations. The application is generated as separate user interface with a menu to access all the different dialogs of the application. (An example of a UIML file is presented in Appendix B) The java code is also generated and contains the connection to the COMDEF services through the use of the user service as well as the UIML renderer to display the user interfaces. Figure 9.8 display the initial menu that is displayed when the client application is started.

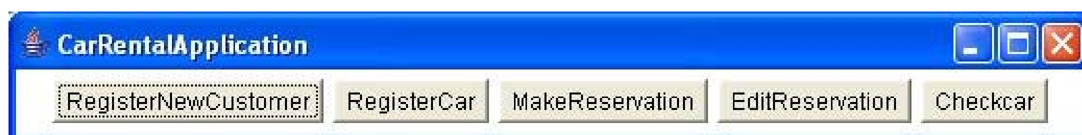

Figure 9.8 – The initial menu of the client

When the user want to open the Register new customer dialog the UIML file contain a mapping of the user interface event and the corresponding action in the client code. An example of this is provided below:

The following code is part of the button definition for the menu button to access the register new customer.

```
<Part name="RegisterNewCustomer" class="button">
  <Style>
    <property name="rendering">Button</property>
    <property name="label">RegisterNewCustomer</property>
  </Style>
</Part>
```

Figure 9.9 – UIML definition of a button.

The following code is an example of how the event triggered in the user interface is defined and mapped to an operation in the client code.

```
<behavior>
  <rule>
    <condition>
      <event class="actionPerformed" part
      name="RegisterNewCustomer"/>
    </condition>
    <action>
      <call name="client.showRegisterNewCustomer"/>
    </action>
  </rule>
</behavior>
(***)
  <peers>
    <logic>
      <d-component name="client" maps-to="client">
        <d-method name="showRegisterNewCustomer" maps-
         to="showRegisterNewCustomer"/>
     </d-component>
    </logic>
   </peers>
```

Figure 9.10 – UIML example of behavior definition.

When the user clicks the button to display the Register new Customer dialog the renderer will call the showRegisterNewCustomer operation in the client which in turn calls the renderer to display this interface. An example of this interface is depicted in figure 9.11 below.

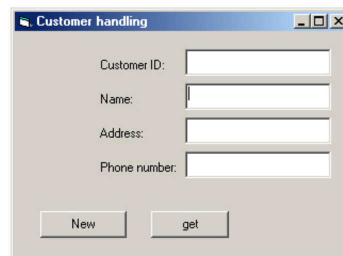

Figure 9.11 - the new customer interface example

An example of the operation called by the renderer to display the user interface is provided below:

```
public void showRegisterNewCustomer()
{
  boolean renderOk = r[0].renderUIML(
    "RegisterNewCustomer.uiml",
    null,//interfaceName
    null,//structureName
    null,//contentName
    null,//styleName
    null,//behavorName
    null,//peersName
    false,// -Dtime option
    false,// -Dnorender option
    false,// -Delements option
    false,// -Dtrace option
    false// -validate option
  );
(***)
  if (renderOk) {
    RegisterNewCustomer_newCustomer_id = (JTextField)
     r[0].getPartByName("newCustomer_id");
    newCustomer_id_item = (JTextField) r[0].getPartByName("id_item");
    newCustomer_id_item = (JTextField) r[0].getPartByName("id_item");
    RegisterNewCustomer_newCustomer_name = (JTextField)
     r[0].getPartByName("newCustomer_name");
    RegisterNewCustomer_newCustomer_adress = (JTextField)
     r[0].getPartByName("newCustomer_adress");
    RegisterNewCustomer_newCustomer_phoneNumber = (JTextField)
      r[0].getPartByName("newCustomer_phoneNumber");
  }

(***)
 }
```

Figure 9.12 – Java code calling the renderer.

If the renderer is able to display the user interface without error, the renderer binds the different UI types to internal variables for further processing.

9.3 Relationship to the distributed system

As described in [Kvalheim, 1999] the userservice is the client interface to the services offered by the distributed COMDEF based system. The userservice package provided by the COMDEF code generator makes it transparent to the client what kind of architecture the underlying system is based on (CORBA, EJB etc.) How much of the user service that is code generated depends on how the service and userservice are linked in the COMDEF. If the user service linked to the service/services with a *provides* relationship the whole userservice is code generated (the userservice maps the service one to one). If the userservice is linked to the service as *require*, the user service need to be implemented post code generations time, as it might require some sort of non trivial processing of data between the client – userservice and userservice-service relationship. In the latter the code generation of the client can only be partial.

9.4 Summary

In this chapter, the case from chapter 2 was revisited and Multidev was applied to the case presented in Chapter 1. The UML model of the Car rental case study was extended with a model of the user interface and the various stages in the Multidev framework was explained in relation to the case study.

Chapter 10

Conclusions, further work and shortcomings

"In theory there is no difference between theory and practice. In practice there is. "
Yogi Berra

This chapter presents the conclusion from this thesis. As part of the conclusion the requirements from chapter 3 are evaluated against the proposed framework. In addition shortcomings are identified and suggestion for further work is presented.

10.1 Evaluation of Multidev against requirements of chapter 3

In this section, Multidev will be evaluated against the requirements of chapter 3. As in Chapter 4, the requirements are rated with the following criteria's:

0 – Not supported
1 – Partially supported
2 – Fully supported

10.1.1 Productivity

Category	Requirement	Support
Code generation	**P1** - Code generate the generic code **P2** - Code generate user interface **P3** - Code generate from graphical notation	**2** **2** **2**
Independency	**P4** - Model environment independent	**2**
General	**P5** - Graphic editable model of GUI **P6** - Template support **P7 -** Developer control **P8** - Support for third party gui components **P9** - Support interface guidelines	**2** **0** **1** **1** **1**

The Multidev framework generates the code to communicate with the interface through the renderer and the DIS services through the user service (P1). In fact, Multidev look at the possibility of using task models to make the client code even more complete, however this work is not completed in this version of Multidev.

The user interface is code generated through the use of UIML (P2) and the framework allows for full generation of the interface with the use of the stereotypes in the metamodel.

The client is generated from a graphical notation in two tiers. Firstly, an overview of the dialogs needed is modeled in the COMDEF UML model. Secondly the tool provided by the framework allows for further refinement of the user interfaces through the use of the three models, the domain, task and presentation model (P3, P5).

This version of Multidev supports XMI indirectly trough a script that converts the XMI to CML. The reason for this approach was elaborated in chapter 6.4. The use of XMI allows Multidev to use all UML modeling tools that support XMI, and hence fulfill the requirement for Model environment independence (P4)

The tool proposed in this framework allows the user to drag and drop between the models used by the tool. This enables the developer to fast set up the user interface parts, and connect the user service through the task model. The models are presented as tree views. This technique was chosen enable the developer to model the user interface fast. The objects in the tree views (nodes and braches) can have their properties set by double clicking on them. (This is described in chapter 8.) (P7)

The framework generates the user interfaces and after system generations it is not trivial to modify these. It should however be quite easy to change and control the user interfaces using the proposed modeling tool. The client application logic can be changed after code generation, and the generated code will generate interfaces to both the user interface and DIS through the user service, that enables the programmer to modify the application logic in an easy manner. (P7)

The Multidev framework Dialog metamodel provides objects that represent standard gui components. These could be used to provide gui templates. As an example a gui could be presented using tabbed panes or a menu that display individual user interfaces. The current version of Multidev does not provide functionality to select among different templates, but this could be added using the objects presented in the *dialog metamodel* , presented in chapter 7.3. (P6)

The purpose of interface guidelines is to ensure applications look right, behave properly, and fit into the intended context. The proposed framework of this thesis consists of well known user interface constructs and this makes user interfaces developed with Multidev somewhat automatically conform to design guidelines. However the framework does not provide any guidelines that consistently and intelligently are used throughout the Multidev development process. The use of templates to suggest user interface design would help to achieve a higher score on the user interface guideline requirement. (P9)

Third party gui components are supported in the dialog metamodel, but it is not trivial to use, as this would require that the gui component must be defined in the UIML language and supported by the various renderers. (P8)

10.1.2 Model concept requirements

Category	**Requirement**	**Support**
General	**M1** - Model based framework **M2** - Understand and use the COMDEF metamodel/ UML profile **M3** – Declarative models	**2** **2** **1**
Architecture	**M4** – User interface and application logic in separate tiers	**2**

Multidev is a model based framework on different levels. Firstly, it uses the COMDEF framework to generate the domain model and extends it to allow for high level of the user interfaces to be used. (M2). Secondly, the Multidev framework uses three models do define the client; the domain, task and presentation model. (M1).

The domain model is generated from the COMDEF model of the DIS system and the Multidev framework introduces a dialog metamodel to be used in the development with the framework. However, the task model in not complete. (M3)

The interface and application logic are in separated tiers in Multidev. The user interfaces are defined in the top tier using UIML and liked to the client application logic through the renderer (M4)

10.1.3 Multi platform support

Category	Requirements	Support
General	**MPS1** – Multi platform support	**1**

Multiplatform support is achieved through the use of UIML, but also limited by this as supported platforms only are those for which a suitable renderer exists. The current version of the tools only support java, but the framework allows for multiple target platforms. (MPS1)

10.1.4 Evolution

Category	Requirement	Support
Evolution	**E1** – Support changes in an easy manner	**1**

The evolution requirement is only partially supported. Changes in the COMDEF model of the system might completely change the domain model and hence leave the presentation (and task model) out of sync. However the tool provided with Multidev allow for rapid regeneration of the client. In addition, if the client logic is programmed by hand, changes to the Multidev models might force reprogramming. (E1)

10.2 Conclusions

The proposed framework presented in this thesis serves as a proposal on how to do rapid development of clients in a distributed environment, based on the COMDEF framework. As this work has been highly experimental, I will not state that this is the only, nor best solution to this problem domain. However this thesis show one direction towards a solution, but more work should be done to be able to evaluate its usefulness in real life case studies. In the next section identified shortcomings are discussed with suggestions for further work.

The evaluation of the requirements in the previous chapter showed that Multidev did not get the maximum score on all requirements. As argued in the previous section, some of the requirements could have achieved a higher score within the constraints of the proposed framework, but where limited because of time constraints.

Although the main task of this work has been to add value to the use of COMDEF, by allowing client and user interface generation, it has also been a focus to generate user interfaces as easy and completely as possible. In this regard Multidev shares some shortcoming with other automatic user interface generations tool. Two of these are worth mentioning; the post editing problem and the matureness of automatic generated user interfaces.
The post editing problem refers to the problem of refinement after the generation of the user interface. Such refinement will either make the original model obsolete, or the refinements will be lost if the user interface is regenerated.

Generated user interfaces have been criticized for being too simple and not mature enough compared to those that are manually coded [Szekely, 1996] [Puerta, 1996]. Multidev share this shortcoming by only being able to use common user interface widgets such as textboxes, labels and so on. Luckily more and more user interface widgets are being standardized and can be used in automatic generation to generate more advances interfaces. In addition the different platforms Multidev can support will increase implicitly, whenever new UIML renderers emerge.

On a more general term a conclusion of the work has been that it is important to take into account the rapid development of the client as well as the distributed system itself. The fast and ever growing use of networks, and the appliances connected to them builds the business case for rapid development of clients and their user interfaces. It seems (from my own experience) the main focus for most projects regarding model development is on the framework for the backend itself, and that the client and user interface is lagging somewhat behind. Although this approach is understandable in regard to resource use, it seem vice to establish a framework that can offer tools for all stages of software development, from modeling, through multiplatform code ,client and user interface generation.

A final conclusion has been that the task of generating user interfaces is enormously complex and various parts of Multidev could be investigated further. As the work on this thesis has revealed to the author, the separate objectives described would be suitable as thesis on their own.

The source code for the Multidev application, this document and other information in regards to this thesis will be available from www.lunde.cc. The various COMDEF code and documents are not available on this website and an inquiry should be made to SINTEF for such information.

10.3 Further work and shortcomings

This section identifies shortcomings to the proposed framework, and suggests paths for further work.

10.3.1 Inherited COMDEF shortcomings

The task of the userservice is to serve as glue between the client application and the underlying middleware. It should be transparent for the client what kind of architecture that is used as middleware. To do this the userservice must deliver plain java objects to the client. This is, however, not the case for the current version of COMDEF. The objects delivered to the client are of the same type as the middleware used. This means that if the DCP is CORBA then the userservice will feed the client CORBA objects, and the client has to import the CORBA packages in order to be able to work with them.

If the methods map one to one with the methods in the service then all of the userservice can be generated from the model in any other case the user services has to be implemented by hand. At many times the client do not need all the information stored in an entity to fulfill its tasks. There might be different services that serve different purposes. As an example, consider the car rental system. There could be a lot more information stored in the car entities such as repair and maintenance intervals. This information would not be of interest to the client application that deals with renting the cars to customers, it would be sufficient for this application to know if the car is free for rental or not. The services might feed the userservices with an object that is only a subset of the entity, only the information the userservice need in order to serve its client. The userservice must cast this object into a plain java (or other platform) specific object type before it delivers it to the client in order for the client to be blind in regards to what middleware that is being used.

The solution to this problem is to let the userservice have a description of the objects in the model. This way, when generating the client the only thing the generator has to worry about is the userservice and the description of the objects it can except and deliver. The way COMDEF is today the client framework has to look at the entities in order to know the

object description of the objects it will deliver and receive from and to the userservice. One way to attain this object model could be to model the userservice in a separate UML model.

10.3.2 Task model

In regard to shortcomings, the implementation of the task model is not complete in this version of Multidev, and would need more work to be complete and fully workable. In addition the framework should be able to let the user select if the task model should be bypassed when generating the client, as at least two reasons for this was identified. Firstly, if the user service is modeled with a require relationship to the services, the userservice is not fully implemented and hence can not serve as a fully qualified model for the client. Secondly, the generation of the application logic can only be a simple passing of information back and forth between the user interfaces and to the underlying DIS. If the client need to perform more complicated operations, the total client generation is difficult.

10.3.3 Wider platform support

The current Multidev emitter can only produce java code and other platforms should be included.

10.3.4 Extending the COMDEF UML profile

In this version of Multidev the COMDEF UML profile was extended with new stereo types to be able to include an overview of the various user interfaces to of the client. The UML profile could be further extended to include a set of user interface stereo types to be able to model the user interfaces more completely in the COMDEF Model. In the recent years interesting work has been done in this regard, such as [Pinheiro da Silva, Paton 2000 #1] and these approaches might be useful for the problem domain covered in this thesis.

10.3.5 Multidev Modeling tool

The modeling tolls developed as a part of this thesis was not developed to be a tool ready to be used by users of the framework, but was built as tool to test the various parts of the framework. All parts of the tool would need to be more finely implemented if the tool was to be used in software development.
A next generation of Multidev should also include sanity check of the GUI Model and between the GUI model and underlying model of tasks and user services (domain model).

10.3.6 Possible Multidev evolvement - COMDEF

Multidev is designed to compliment the COMDEF framework with the ability to simplify the process of developing clients that can utilize this framework. As stated in chapter 5, the development of COMDEF has ended. COMET [comet, 2006] can be viewed as the next generation of COMDEF. Therefore a possibility for further work is to adjust Multidev to fit within the boundaries of the COMET framework. As an example of an adjustment that would be needed is that the CML language would not longer be used for client generation. The basis for generation for Multidev within the COMET framework would be the UML model, described in XMI. This would not be a problem, as the UML, CML and XMI can be mapped one to one from one to another.

10.3.7 Possible Multidev evolvement - UIML

UIML was chosen as the platform user interfaces because of the ability to utilize the developed interfaces on platform of different thickness. Other technologies has emerged that could compete with UIML in this regard. Two of these are AUIML [Azevedo et al. 2000] and USIXML [Limbourg et al., 2004]

As stated in chapter 5, UIML is still actively developed and new rendereres have emerged. At the time of writing there are 11 different platforms [Harmonia, 2006] for which there are

renderers. This implies that a new version of Multidev would have a wide range of platforms that could be targeted, and hence is still a good candidate.

A new version of Multidev should seek to have more of the user interfaces modeled in the UML model. This can be done by more extended usage of the dialog metamodel presented in this thesis. I would also suggest that a futuristic version of Multidev should seek to become more compliant with the MDA framework. This chapter shows that there are some similarities and these should be explored further. Especially the similarities between Multidev and MDA are interesting in the context of a new generation of Multidev to be used in COMET.

10.3.8 Latest Model Based Developments and Multidev

COMDEF was an early approach towards model based development. Since COMDEF, other frameworks have emerged. At the time of writing the most eminent of these are:

- MDA - Model Based Architecture from OMG
- DSL – Domain Specific Languages from Microsoft.

The most significant factor contributing to the growing number of frameworks is the broadened understanding of how to develop metamodel based frameworks. These recent developments can be compared to automation of compiler technology in earlier times, such as lex [Wikipedia, lex 2006] and [Wikipedia, Yacc 2006]. Now the same level of automation can be achieved for domain specific languages.

The focus of MULTIDEV is independent of the actual metamodel framework that is being used. Using the experiences from COMDEF as a base, the framework presented in this thesis can be transferred to other similar frameworks, such as the OMG DMA and Microsoft DSL.

This section discusses the directions taken by the two leading paths within the world of model based development, mentioned above. These two initiatives will be discussed in relation to the exiting Multidev framework and how these frameworks can be used by future versions of Multidev.

10.3.9 OMG and MDA

Model Driven Architecture (MDA) [Kleppe et al. 2003] is an initiative started by the OMG [OMG 2006] in 2000. According to [Kleppe et al. 2003] the primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns. To better understand this statement the next section discusses the basic concepts of MDA. The various concepts are not discussed in details. Please look at [Kleppe et al. 2003], [Mellor et al. 2004] for further reference.

10.3.9.1 Basic concepts of MDA

In the following the basic concepts of MDA are discussed. For further reference please see [Kleppe et al. 2003]

10.3.9.1.1 Productivity and reusability

As written above, part of the MDA promise is to increase productivity and reusability in the software development process. Partly this is achieved by shifting focus from traditional specifications and coding to modeling of the platform independent models (PIM; see below). This focus shift allows developers to focus on the business tasks of the software and not the details of the various platforms.
The transformations from PIM to Platform Specific Model (PSM; see below) and then to actual running software can be reused within the boundaries of the MDA.

10.3.9.1.2 Portability

Again one of the MDA promises is portability. In MDA this is achieved through the transformation from one PIM to several PSM. This implies that once the PIM is modeled you can transform the PIM to all PSM where there are transformation tools available.

10.3.9.1.3 Interoperability

Within the MDA framework interoperability is achieved through *bridges* between different models. A PIM may is typically transformed into one or several PSM. An example is generating both a PSM for Java code and a PSM for a relational database. This will allow us to map a Java Object to its equivalent stored information in a data base.

10.3.9.1.4 Maintenance and documentation

As noted above the MDA shifts developers focus from traditional tasks to the task of modeling PIM. The PIM is used to generate PSM which in turn is used to generate the source code. This means that the PIM is both a formal representation of the software, but also serves as high level documentation for the system it represents. Today modeling is often used as a tool in software development, but the model is often abandoned when the actual coding starts, or at least when the system is implemented. When system changes or maintenance is needed the model is often not updated. In an MDA environment this is changed dramatically. Here system changes and maintenance is done by changing the PIM.

10.3.9.1.5 Platform

The definition of platforms varies greatly. [Mellor et al. 2004] defines a platform as the specification of an execution environment for a set of models.

10.3.9.1.6 Architecture

Architecture is a popular term in software development. The term is borrowed from the art and science of designing buildings and structures [Wikipedia Architecture 2006]. Both the traditional architect and the software architect will use models as an important tool for designing building or software. [Murkerji, Miller 2003] defines architecture as the parts and connectors of a system as well as the rules for interaction using the connectors. MDA describes a set of different models and the role and relationship between these models.

10.3.9.1.7 Model

A model is a word that has different meaning depending on the context where it is used. In the context of MDA a model is description of a software system.

10.3.9.1.8 Model-driven

By the term model-driven it is implied an approach to development. In this approach models are the primary tool for the development process. Models are used is used throughout the phases of development. These phases include (but are not limited to) analysis, design, implementation, documentation and maintenance.

10.3.9.1.9 Views and viewpoints

A viewpoint is a position from which something is observed or considered. Applying a viewpoint to a system will let you observe the system from a level of abstraction defined by a set of architectural concepts and structuring rules. A view is the representation of a system through a certain viewpoint.

MDA provides 3 viewpoints, namely computational independent, platform independent and platform specific. These viewpoints are realized through the use of models.

10.3.9.1.10 Computational Independent Model (CIM)
The Computational Independent Model (CIM) is used to model business processes, stakeholders, departments and so on. A CIM can also be referred to as a business model and does not necessarily describe any part of the software system.

10.3.9.1.11 Platform Independent Models (PIM)
The Platform Independent Model is to be the entry point for software developers in the MDA framework. The PIM is a view of the system at an abstraction level where, to some degree, the platform specific constraints are not considered. This partly platform independence will let the system be suitable multiple platforms. One of the benefits of developing software at this level of abstraction is that the developer can focus on the business processes rather then platform specific details.

10.3.9.1.12 Platform Specific Models (PSM)
The Platform Specific Model (PSM) is a viewpoint in MDA that resides at a level of abstraction where the details of the selected target platform or platforms are known. A PSM is usually the result of a transformation of one or more PIMs.

10.3.9.1.13 Transformation
In MDA a transformation is the process of converting a model of a system to another model of the system. A PSM is usually a transformation from one or more PIM. Another example is converting a PSM to another PSM.

10.3.10 Software Factories and Microsoft Domain Specific Language
Microsoft is stepping up to the challenge and is expanding their development flagship Visual Studio with new features in the area of model driven development. In addition Microsoft is pursuing an initiative called Software Factories [Greenfield et al., 2004] which may be seen as a competitor to the MDA framework for Model Driven Software Development. The Microsoft initiative is more focused on efficiency then on portability [Bråthen, 2005].

10.3.10.1 Basic concepts
In the following the basic concepts of Software Factories and Domain Specific Languages are discussed, albeit not in detail. For further reference please see [Greenfield 2004] and [MS Soft. fac., 2006].

10.3.10.1.1 Software Factories
A modern factory has assembly lines that be configured to manufacture different versions of the same product. The most classic example is an assembly line producing cars. One assembly line can assemble different car models through configuration changes to the line. The idea of Software Factories is to utilize this already proven technology for productive manufacturing into the world of software. According to [Greenfield 2004] Software Factories are based on three key ideas; software factory schemas, software factory template and an extensible development environment.

10.3.10.1.2 Software Factory Schema (SFS)
[Greenfield et al., 2004] states that a Software Factory Schema describes the artifacts that must be developed to produce a software product. Artifacts in this context mean XML documents, models, configuration files source code, SQL files. The Software Factory Schema is used to store all of these artifacts in a systematic manner. In addition the schemas are used to define relationships between the artifacts. Software schemas are most often represented as either a grid or a graph. In the grid view the rows define a level of abstraction while the columns define concern. The cell is to be considered as a viewpoint from which it is possible to build some aspect of the software. [Greenfield et al., 2004] states that a grid is

to be considered as an abstraction of the more accurate representation of the SFS, namely the representation as a graph. In this representation the nodes are viewpoints and the edges are computable edges between the viewpoints.

The Software Factory Schema is built to accommodate the development of a certain family of software. In order to be used, the schema must be configured for a certain software product.

10.3.10.1.3 Software Factory Template

The software factory template is the implementation of the artifacts described by the Software Factory Schema. The implemented artifacts can be loaded into an Interactive Development Environment (IDE) and thus configure this to build a certain software product. An analog procedure is the use of different document templates in MS Word to produce a certain type of document.

10.3.10.1.4 Extensible Development Environment

An Extensible Development Environment is the actual Software Factory that in the end produces the software. This development environment will be configured using the Software Factory Template, which consists of different artifacts as described above. Microsoft has incorporated some of the features needed to build Software Factories in to its development flagship, Microsoft Visual Studio 5.0 [Microsoft, 2003]

10.3.10.1.5 Domain Specific Language (DSL)

A Domain Specific Language (DSL) enables the specification of software from a specific viewpoint [Greenfield 2004]. A DSL is designed to be useful for a specific task in a defined problem domain. Examples of where DSL can be used are many, and include description of a user interface, a business process, a database, or the flow of information. The description can then be used to generate code. DSL can be either textual or graphical in its notation.

Microsoft is supporting the idea of Software Factories and Domain Specific Languages in the software development product called Visual Studio. A recently added feature to this product is called Domain Model Designer. This feature is designed to let the developer design DSL and use these in software development.

10.3.11 Multidev and MBD’s new approaches

The differences between the approaches above are many. However, at high level of abstraction they strive to achieve the same; efficient software development through the use of models and code generation. Their similarities and differences are discussed else ware such as in [Bråthen 2005] and [Greenfield 2004].

This section look at some of the similarities of the Multidev Framework described in this thesis and the approaches of the above introduced MDD initiatives. Since the two approaches discussed above seem to play leading roles in the model based development sphere, it is interesting to see that there are some similarities in the Multidev methodology and the once used by these initiatives.

10.3.11.1 Multidev and MDA

Multidev utilizes tools and languages that have same origin as MDA, namely the OMG. At the bases is the modeling language UML which in Multidev is used to model user interfaces. This section will discuss the similarities between the MDA and the Multidev Framework.

A Platform Independent Model (PIM) is at the heart of the MDA. This is where software developing will reside. The Multidev framework presents an extension to the COMDEF UML profile that enables the developer to model the layout of the dialogs needed in the COMDEF application. Although this model is very simple it can be viewed as a PIM as this model, containing the dialogs, do not have any platform specific details.

Continuing to look at the dialog model as a PIM we see that Multidev also allows for a transformation from one model to another through the use of the emitter. The resulting model is used as an input into the Multidev modeling tool. This second stage model can be viewed as both a PIM and a PSM. It is a PIM in the sense that it is still platform independent, because at this point there is no limitation that binds it to a specific platform. This is done at the later stage, at the time when the model is transformed into UIML files.

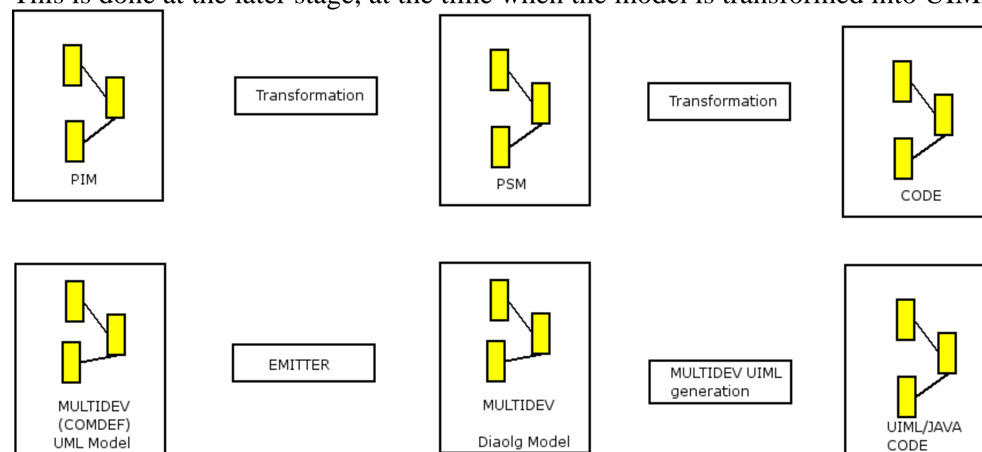

Figure 10.1 - MDA Multidev process similarities.

There is however a breakage in the Multidev, PIM and PSM similarities. When the UML Model is transformed through the emitter and moved into the Multidev modeling tool, the model may be refined further. This means that the model can not be transformed back into the PIM, which breaks the analogy.

10.3.11.2 Multidev and Software Factories + DSL

The similarities between Multidev and Software Factories are not as visible as with MDA. However some similarities are found. The COMDEF and Multidev framework can be viewed as a software factory for creating distributed systems. As described earlier a high level view of a Software Factory reveals three main parts; software factory schemas, software factory template and an extensible development environment. All UML models, the CML files, emitters, the Multidev modeling tool and the different generated files can be viewed as the Software Factory Template. The COMDEF and Multidev framework can be viewed as the Software Factory Schema (SFS). There is no extensible development environment to configure with the SFS, instead there are custom tools that provide the same functionality.

Part of a Software Factory Schema is Domain Specific Languages. The concept of Domain Specific Language can be compared to the Multidev Dialog metamodel, describe in chapter 7. This metamodel can be used to model user interfaces and nothing else. It is sharply limited to this problem domain and may be viewed as a Domain Specific Language for the modeling of user interfaces.

10.4 Summary

This chapter has contains the conclusions of the work done in this thesis, and high level investigation of the shortcomings found. The main conclusion is that the selected approach could serve as a tool for rapid development of clients utilizing a distributed environment. This chapter has also looked at how the work presented in this thesis could be extended on its own and in combination with COMET. In addition this chapter included overview on two initiatives in Model Driven Development. The current framework of Multidev was discussed in reference to these technologies.

Appendix A

The Car Rental System and COMDEF

This Appendix gives a short introduction into how the Car Rental system is modeled in COMDEF. For a more complete understanding of the COMDEF framework, please refer to [Kvalheim 99]. This short introduction focuses on the system from a COMDEF viewpoint, and does not consider the Multidev features described in this thesis.

The Car rental system is an example implementation of system for processing rental. Since there is no need for a fully featured system, the system is modeled with only a few basic features.

The figure below describes the COMDEF development process.

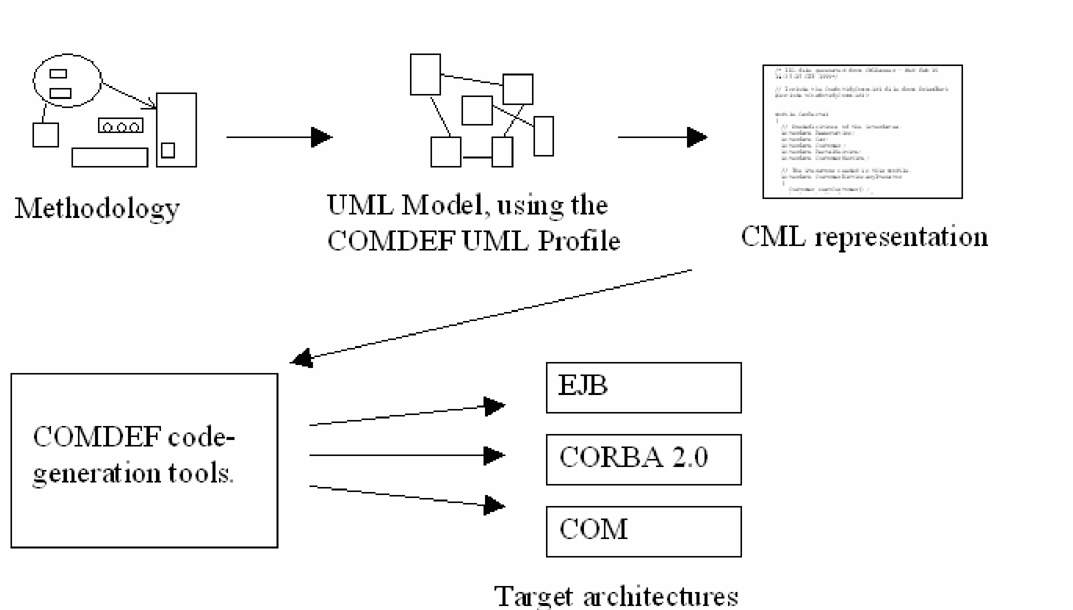

Figure A1 – The COMDEF development process

The methodology used to build the Car Rental system goes through all of the steps in the figure above. This is however not required for all COMDEF methodologies as the UML part may be skipped if CML code can be generated without the use of UML. (The UML and CML representations of the system maps one to one)

The Car Rental system was developed using a case driven methodology. Use cases are used to capture the requirements of a system. 8 use cases are the basis for the Car Rental system. These are described in chapter 2.1.

When all use cases where identified the system was modeled in UML using the COMDEF UML Profile. The Model of the system is depicted in Figure 2.1 in chapter 2. The system contains the following elements:

- **1 User Services**
  Car rental service

- **2 Business Events**
  Not Checked in
  Not Checked Out

- **3 Entities**
  Car
  Customer
  Reservation

- **2 services**
  Customer Service
  Rental service

When the model was ready it was translated into CML. This is done through a UML2CML script. The following depicts a CML example of an entity from the Car Rental Model. The complete CML file and the UML2CML script can be downloaded from http://www.lunde.cc.

```
// A Reservation has, a list of cars reserved and a relation to
the customer If a car
// is not checked in or out the appropriate event is sent.
entity Reservation
{
    relationship  cars;
    relationship Customer customer
        inverse Customer::reservations;

    attribute long fromDate;
    attribute long toDate;
    attribute long refId;
    attribute long actualOutDate;
    attribute long actualInDate;

    signal NotCheckedIn;
    signal NotCheckedOut;

    string status();
};
```

Figure A2 – A CML Entity example

The CML is the starting point for platform specific code generation in COMDEF using the COMDEF Code Generation Tools. The code generation tool is called an emitter. The corresponding java code for the Reservation entity is depicted below. (Some code ha been omitted in the example below.

```
package entities;
. . .
public class Reservation
implements Serializable {

    private Date fromDate = null;
    private Date toDate= null;
    private long refId;
    private Date actualOutDate = null;
    private Date actualInDate = null;
    private Customer customer = null;
    private Car car = null;

    public Reservation(Date fromDate, Date toDate,long refId) {
      this.fromDate = fromDate;
      this.toDate = toDate;
      this.refId = refId;
    }

    public void setFromDate(Date fromDate) {
       this.fromDate = fromDate;
    }

    public Date getfromDate() {
       return fromDate;
    }

    public void setToDate(Date toDate) {
        this.toDate = toDate;
    }

    public Date getToDate() {
       return toDate;
    }

    public void setRefId(long refId) {
       this.refId = refId;
    }

    public long getRefId() {
       return refId;
    }

    public void setActualOutDate(Date actualOutDate) {
       this.actualOutDate = actualOutDate;
    }

   public Date getActualOutDate() {
      return actualOutDate;
    }
   . . .
    public void setCar(Car car) {

      this.car = car;
    }

       public Car getCar() {

      return car;
    }

    // --------- Serializable --------------
   private void writeObject(java.io.ObjectOutputStream out) throws IOException {
     out.defaultWriteObject();
   }

   private void readObject(java.io.ObjectInputStream in)
     throws IOException, ClassNotFoundException {
     in.defaultReadObject();
   }

}
```

Figure A3 – Generated Java code for Reservation Entity

The generated generic code servers as the basis for client development.

After the code-generation of the generic code, the implementation of the different methods defined in the system model is left for the developer. This effort should be easy as the developer can use the code-generated API to access the attributes, the relationships.

All the distribution and persistent storage is done by COMDEF, the underlying database and the DCP.

Appendix B

An UIML example file

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML 2.0 Draft//EN" "UIML2_0d.dtd">


<!-- UIML file generated from UIML tool - Tue Dec 05 11:34:25 GMT+01:00 2000-->

<uiml xmlns='http://uiml.org/dtds/UIML2_0a.dtd'>
  <head>
  </head>
  <interface name="CustomerRegistration">
    <STRUCTURE>
      <Part name="CustomerRegistration" class="frame">
        <Style>
          <property name="rendering">Frame</property>
          <property name="title">Customer Registration</property>
          <property name="layout">null</property>
          <property name="resizable">true</property>
          <property name="location">50,50</property>
          <property name="size">250,200</property>
        </Style>
        <Part name="customerIdLabel" class="label">
       <Style>
         <property name="rendering">Label</property>
         <property name="text">Customer id:</property>
         <property name="location">15,25</property>
            <property name="size">75,20</property>
       </Style>
        </Part>
        <Part name="customerIdText" class="Text">
          <Style>
            <property name="rendering">JTextField</property>
            <property name="columns">10</property>
            <property name="location">115,25</property>
            <property name="size">100,20</property>
          </Style>
        </Part>
        <Part name="customerNameLabel" class="label">
          <Style>
            <property name="rendering">Label</property>
            <property name="text">Customer name:</property>
            <property name="location">15,50</property>
            <property name="size">100,20</property>
          </Style>
        </Part>
        <Part name="customerNameText" class="Text">
          <Style>
            <property name="rendering">JTextField</property>
            <property name="columns">10</property>
```

```
        <property name="location">115,50</property>
        <property name="size">100,20</property>
      </Style>
    </Part>
    <Part name="customerAdressLabel" class="label">
      <Style>
        <property name="rendering">Label</property>
        <property name="text">Adress:</property>
        <property name="location">15,75</property>
        <property name="size">100,20</property>
      </Style>
    </Part>
    <Part name="customerAdressText" class="Text">
      <Style>
        <property name="rendering">JTextField</property>
        <property name="columns">10</property>
        <property name="location">115,75</property>
        <property name="size">100,20</property>
      </Style>
    </Part>
    <Part name="customerPhoneNumberLabel" class="label">
      <Style>
        <property name="rendering">Label</property>
        <property name="text">Phone number:</property>
        <property name="location">15,100</property>
        <property name="size">100,20</property>
      </Style>
    </Part>
    <Part name="customerPhoneNumberText" class="Text">
      <Style>
        <property name="rendering">JTextField</property>
        <property name="columns">10</property>
        <property name="location">115,100</property>
        <property name="size">100,20</property>
      </Style>
    </Part>
    <Part name="okButton" class="button">
      <Style>
        <property name="rendering">Button</property>
        <property name="label">Ok</property>
        <property name="location">80,150</property>
        <property name="size">50,20</property>
      </Style>
    </Part>
    <Part name="CancelButton" class="button">
      <Style>
        <property name="rendering">Button</property>
        <property name="label">Cancel</property>
        <property name="location">135,150</property>
        <property name="size">50,20</property>
      </Style>
    </Part>
  </Part>
</STRUCTURE>
```

```
    <behavior>
      <rule>
        <condition>
          <event class="actionPerformed" part-name="okButton"/>
        </condition>
        <action>
          <call name="client.newCustomer"/>
        </action>
      </rule>
    </behavior>

  </interface>
  <peers>
  <logic>
    <d-component name="client" maps-to="client">
    <d-method name="newCustomer" maps-to="newCustomer"/>
    </d-component>
  </logic>

  </peers>
</uiml>
```

Appendix C

The Complete CML Grammar

The grammar is shown with Extended Backus Naur Form (EBNF). The grammar includes the Multidev extensions. Keywords are shown in bold font while the non-terminals are shown in regular font. Definitions in upper case and between <> are used to indicate words defined in the lexical part of the grammar.

```
specification       ::=  ( definition )+
definition          ::=  module ";"
                         | userservice_dcl ";"
                         | service_dcl ";"
                         | event_dcl ";"
                         | entity_dcl ";"
                         | interface_dcl ";"
                         | const_dcl ";"
                         | type_dcl ";"
                         | except_dcl ";"
                         | application_dcl ";"
                         | dialog_dcl ";"
module              ::=  module identifier "{" ( definition )* "}"
interface_dcl       ::=  interface identifier ( ( inheritance_spec )?
                         "{" interface_body "}" )?
inheritance_spec    ::=  ":" scoped_name ( "," scoped_name )*
interface_body      ::=  ( export )*
export              ::=  type_dcl ";"
                         | const_dcl ";"
                         | except_dcl ";"
                         | attr_dcl ";"
                         | op_dcl ";"
userservice_dcl     ::=  userservice identifier ( inheritance_spec )?
                         ( "{" userservicebody_dcl "}" )?
userservicebody_dcl ::=  ( export_userservice )*
export_userservice  ::=  signal_dcl ";"
                         | subscribe_dcl ";"
                         | op_dcl ";"
                         | requires_dcl ";"
                         | provides_dcl ";"
service_dcl         ::=  service identifier ( inheritance_spec )?
                         ( "{" servicebody_dcl "}" )?
servicebody_dcl     ::=  ( export_service )*
export_service      ::=  type_dcl ";"
                         | const_dcl ";"
                         | except_dcl ";"
                         | attr_dcl ";"
                         | op_dcl ";"
                         | signal_dcl ";"
                         | subscribe_dcl ";"
                         | relationship_dcl ";"
                         | requires_dcl ";"
                         | provides_dcl ";"
event_dcl           ::=  event identifier "{" eventbody_dcl "}"
eventbody_dcl       ::=  ( eventexport )*
eventexport         ::=  event_attr_dcl ";"
entity_dcl          ::=  entity identifier ( inheritance_spec )?
                         ( "{" entitybody_dcl "}" )?
entitybody_dcl      ::=  ( export_entity )*
export_entity       ::=  type_dcl ";"
                         | const_dcl ";"
                         | except_dcl ";"
                         | attr_dcl ";"
                         | op_dcl ";"
                         | signal_dcl ";"
                         | subscribe_dcl ";"
                         | relationship_dcl ";"
                         | requires_dcl ";"
                         | provides_dcl ";"
```

```
dialog_dcl          ::= dialog identifier("{""}")
application_dcl     ::= application identifier ("{"applicationbody_dcl"}")
applicationbody_dcl ::= (contains_dialog ";")*
contains_dialog     ::= contains scoped_name
attr_dcl            ::=  ( readonly )? ( transient )? ( factory )?
                         attribute param_type_spec ( identifier
                         ( "[" positive_int_const "]" )? | "*" identifier )
event_attr_dcl      ::=  param_type_spec ( identifier
                         ( "[" positive_int_const "]" )? | "*" identifier )
except_dcl          ::=  exception identifier "{" ( member )* "}"
relationship_dcl    ::=  relationship relationship_type identifier
                         ( inverse scoped_name )?
relationship_type   ::=  ( param_type_spec
                         | dictionary "<" param_type_spec "," param_type_spec ">"
                         | set "<" param_type_spec ">"
                         | bag "<" param_type_spec ">"
                         | list "<" param_type_spec ">" )
op_dcl              ::=  ( factory )? ( param_type_spec | void ) identifier
param_dcls
param_dcls          ::=  "(" ( param_dcl ( "," param_dcl )* )? ")"
param_dcl           ::=  param_attribute param_type_spec identifier
param_attribute     ::=  in | out | inout
param_type_spec     ::=  base_type_spec | scoped_name | string_type
requires_dcl        ::=  requires scoped_name ( "," scoped_name )*
provides_dcl        ::=  provides scoped_name ( "," scoped_name )*
signal_dcl          ::=  signal scoped_name
subscribe_dcl       ::=  subscribe scoped_name
scoped_name         ::=  ( "::" )? identifier ( "::" identifier )*
type_dcl            ::=  typedef_dcl | struct_type | union_type | enum_type
typedef_dcl         ::=  typedef type_spec declarators
type_spec           ::=  simple_type_spec
                         | constr_type_spec
simple_type_spec    ::=  base_type_spec
                         | template_type_spec
                         | scoped_name
base_type_spec      ::=  floating_pt_type | integer_type | char_type
                         | boolean_type | octet_type | any_type
template_type_spec  ::=  sequence_type | sring_type
constr_type_spec    ::=  struct_type | union_type | enum_type
declarators         ::=  declarator ( "," declarator )*
declarator          ::=  complex_declarator | simple_declarator
simple_declarator   ::=  identifier
complex_declarator  ::=  array_declarator
floating_pt_type    ::=  float | double
integer_type        ::=  signed_int | unsigned_int
signed_int          ::=  long | short
unsigned_int        ::=  unsigned_long_int | unsigned_short_int
unsigned_long_int   ::=  unsigned long
unsigned_short_int  ::=  unsigned short
char_type           ::=  char
boolean_type        ::=  boolean
octet_type          ::=  octet
any_type            ::=  any
struct_type         ::=  struct identifier "{" member_list "}"
member_list         ::=  ( member )+
member              ::=  type_spec declarators ";"
union_type          ::=  union identifier switch "(" switch_type_spec ")"
                         "{" ( casex )+ "}"
switch_type_spec    ::=  integer_type | char_type | boolean_type
                         | enum_type |scoped_name
casex               ::=  ( case_label )+ element_spec ";"
case_label          ::=  ( case const_exp ":" | <DEFAULTTOK> ":" )
element_spec        ::=  type_spec declarator
enum_type           ::=  enum identifier "{" enumerator ( "," enumerator )* "}"
enumerator          ::=  identifier
sequence_type       ::=  sequence "<" simple_type_spec ( "," positive_int_const )?
">"
string_type         ::=  string ( "<" positive_int_const ">" )?
array_declarator    ::=  identifier ( fixed_array_size )+
fixed_array_size    ::=  "[" positive_int_const "]"
positive_int_const  ::=  const_exp
identifier          ::=  <IDENTIFIER>
const_dcl           ::=  const const_type identifier "=" const_exp
const_type          ::=  integer_type | char_type | boolean_type
                         | floating_pt_type | string_type | scoped_name
```

```
const_exp           ::=  xor_expr ( "|" xor_expr )*
xor_expr            ::=  and_expr ( "^" and_expr )*
and_expr            ::=  shift_expr ( "&" shift_expr )*
shift_expr          ::=  add_expr ( ( ">>" | "<<" ) add_expr )*
add_expr            ::=  mult_expr ( ( "+" | "-" ) mult_expr )*
mult_expr           ::=  unary_expr ( ( "*" | "/" | "%" ) unary_expr )*
unary_expr          ::=  ( unary_operator )? primary_expr
unary_operator      ::=  "-" | "+" | "~"
primary_expr        ::=  scoped_name | literal | "(" const_exp ")"
literal             ::=  integer_literal | string_literal | character_literal
                         | floating_pt_literal | boolean_literal
boolean_literal     ::=  true | false
4integer_literal    ::=  <OCTALINT> | <DECIMALINT> | <HEXADECIMALINT>
string_literal      ::=  <STRINGLIT>
character_literal   ::=  <CHARACTER>
floating_pt_literal ::=  <FLOATONE> | <FLOATTWO
```

Appendix D

Bibliography

[Myers, 1992]	Myers, B.A.; Rosson, M.B.: Survey on User Interface Programming, in CHI 92.195-202.
[World stats, 2006]	Internet World Stats http://www.internetworldstats.com/stats.htm
[Machiraju, 1996]	Vijay Machiraju,A Survey on Research in Graphical User Interfaces,1996
[Microsystems, 2001 #1]	Abstract Window Toolkit - AWT Sun Microsystems http://java.sun.com/j2se/1.4.2/docs/guide/awt/index.html
[Microsystems 2001 #2]	A Swing Architecture Overview, Amy Fowler, http://java.sun.com/products/jfc/tsc/articles/architecture/index.html 1998
[Olsen, 1987]	Dan R. Olsen Jr. "Larger Issues in User Interface Management," Computer Graphics. 1987. 21(2). pp. 134-137.
[Microsoft, 2001]	MFC Microsoft Foundation classes, 2005 http://msdn2.microsoft.com/en-us/library/65dtx4a4.aspx
[Metrowerks, 2001]	CodeWarrior PowerPlant Metrowerks (Freescale) http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=012726
[Symantec,webgain, 2005]	Visual café (WebGain) http://en.wikipedia.org/wiki/Visual_Cafe http://www.symantec.com
[Microsoft,2003 #1]	Microsoft Visual Studio msdn.microsoft.com/vstudio/
[Microsoft,2003 #2]	Microsoft Access, http://www.microsoft.com/office/access/prodinfo/overview.mspx
[Sukaviriya et al., 1993]	Sukaviriya, P., Foley, J. D., and Todd, G. A second generation user interface design environment: The model and the runtime architecture. In Proceedings of INTERCHI'93, pages 375--382, Apr. 1993.
[Szekely et al., 1993]	Beyond Interface Builders: Model-Based Interface Tools. P. Szekely, P. Luo, and R. Neches. In Proceedings of INTERCHI'93 April, 1993, pp. 383-390.
[Neches et al., 1993]	Robert Neches , Jim Foley , Pedro Szekely , Piyawadee Sukaviriya , Ping Luo , Srdjan Kovacevic , Scott Hudson, Knowledgeable development environments using shared design models, Proceedings of the 1st international conference on Intelligent user interfaces, p.63-70, January 04-07, 1993, Orlando, Florida, United States
[Szekely, 1996]	Szekely, P.: Retrospective and Challenges for Model-Based Interface Development. In Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces, (Vanderdonckt, J. Ed.). Namur University Press, Namur, 1996.
[Puerta, 1996]	Puerta, A.: Issues in automatic generation of user interfaces in model-based systems. In: Vanderdonckt, J. (Hrsg.), Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces (CADUI'96) Namur, 5-7 June 1996.
[Kvalheim, 1999]	Bård Kvalheim: COMDEF - a flexible framework for component development, Cand. Scient Thesis, Department of informatics, University of Oslo 1999
[Frankel, 2001]	David S. Frankel, XMI: The OMG's XML Metadata Interchange Standard http://www.sys-con.com/xml/archives/0104/Frankel/index.html 7 februar 2001
[Genera, 2000 #1]	Genera: Developer framework version 6.1 www.genera.no Genera 2000
[Genera, 2000 #2]	Genera: White paper - http://www.genera.no/genova_wp.htm Genera 2000

[Vogt, 2001]	Yngve Vogt: Nordmenn I verdensteten, Computerworld 16 januar 2001 http://www.computerworld.no/index.cfm/fuseaction/artikkel/id/8008
[Puerta, 1997]	Angel R. Puerta: A Model-Based Interface Development Environment , IEEE Software Volume 14,#4 p. 40-47, 1997 IEEE Computer Society Press, http://dx.doi.org/10.1109/52.595902
[Puerta, Einstein 1999]	A. Puearta and J. Einstein: Towards a general computational framework for modelbased interface development systems. In International Conferences on Intelligent User Interfaces, Pages 171-178, January 1999
[Puerta, 1998]	Puerta, A.R. "Supporting User-Centered Design of Adaptive User Interfaces Via Interface Models". First Annual Workshop On Real-Time Intelligent User Interfaces For Decision Support And Information Visualization, San Francisco, January 1998.
[Puerta, Cheng 1999]	A. R. Puerta, E. Cheng, T. Ou, and J. Min. MOBILE: User-centered interface building. In CHI: Human Factors in Computing Systems, pages 426--433. ACM Press, May 1999.
[Oboe, 1999]	Oboe project: Oboe White paper, 1999, www.opengroup.org/oboe
[CORBA 1996]	The Common Object Request Broker: Architecture and Specification Revision 2.0 OMG 1996 http://www.omg.org/docs/formal/97-02-25.pdf
[ODMG,1997]	Object Database Standard: ODMG 2.0, Morgan Kaufmann Publishers, inc, San Fransisco, 1997, ISBN 1-55860-463-4
[Flanagan, 1997]	Flanagan, D: Java in a Nutshell,O'Reilly & Associates Inc.,1997 ,ISBN 1-56592-262-X
[WC3-XML, 1998]	WC3: Extensible Markup Language (XML) 1.0, 1998 http://www.w3.org/TR/1998/REC-xml-19980210
[Maruyama et al. 1999]	Maruyama,Hiroshi Tamura,Kent Uramoto,Naohiko: XML and Java - Developing Web Applications,Addison-Wesley, 1999, ISBN 0-201-48543-5
[OMG XMI, 1999]	OMG: XML Metadata Interchange (XMI) Specification, Version 1.1,1999 - http://cgi.omg.org/cgi-bin/doc?ad/9910-02 http://cgi.omg.org/cgi-bin/doc?ad/9910-03
[WC3-XSLT,1999]	WC3: XSL Transformation (XSLT) Version 1.0 W3C recommendation 16 November 1999 http://www.w3.org/TR/xslt
[WC3-XSL,2001]	The Extensible Stylesheet Language Family (XSL) 2001 - http://www.w3.org/Style/XSL/
[Harmonia,2001]	Harmonia inc.Virginia Tech Corporate Research Center,1715 Pratt Drive,Suite 3100,Blacksburg VA, 24060,USA www.harmonia.com
[Marc, 2001]	Marc Abrahams: UML Projects at Virginia Tech, Computer Science Department, 0106 ,Blacksburg, VA 24061, http://vtopus.cs.vt.edu/~abrams/uiml/ 2001
[UIML 1.0,1997]	Harmonia: UIML reference manual, UIML 1.0 1997 - http://www.uiml.org/specs/docs/uiml_v10_ref.PDF
[Booch et al.,1999]	Booch,Grady Rumbaugh,James Jacobsen,Ivar:The Unified Modeleing Language Users Guide, Addison Wesley Longmann inc.1999 ISBN:0-201-57168-4
[Cockburn, 1997]	Cockburn,Structuring Use cases with goals,Journal of Object-Oriented Programming 1997 Sep-Oct, Nov-Dec, page 16 - http://members.aol.com/acockburn/papers/usecases.htm
[Meggison -tech, 2000]	Megginson Technologies: SAX - Simple API for XML, 2000, http://www.megginson.com/SAX/
[OMG UML, 1999]	OMG: OMG UML v. 1.3 specification, 1999, http://www.rational.com/uml/resources/documentation/index.jsp

[Storrs, 1995] G. Storrs: The Notion of Task in Human-Computer Interaction. In: M. Kirby, A. Dix, J. Finlay (eds.): People and Computers X. Proceedings British HCI'95 (Huddersfield UK, August 1995). Cambridge: Cambridge University Press, 1995,357-365.

[Griffiths et al., 1999] Tony Griffiths, Peter J. Barclay, Jo McKirdy, Norman W. Paton, Philip D. Gray, Jessie Kennedy, Richard Cooper, Carole A. Goble, Adrian West and Michael Smyth, (1999), Teallach: A Model-Based User Interface Development Environment for Object Databases, in Proc. User Interfaces to Data Intensive Systems (UIDIS), IEEE Press. pp. 86-96. 1999. http://citeseer.ist.psu.edu/article/griffiths99teallach.html

[Markopoulos et al., 1992] P. Markopoulos, J. Pycock, S. Wilson, and P. Johnson, "Adept - A task based design environment", Proceedings of the 25th Hawaii International Conference on System Sciences, IEEE Computer Society Press, 1992, pp. 587-596. http://citeseer.ist.psu.edu/markopoulos92adept.html

[Browne et al., 1992] T. P. Browne et al. Using declarative descriptions to model user interfaces with MASTERMIND. In F. Paterno and P. Palanque, editors, Formal Methods in Human Computer Interaction. Springer-Verlag, 1997. http://citeseer.ist.psu.edu/browne97using.html

[Schlungbaum, 1998] CHI'98 Workshop, (Knowledge-based) Support of Task-based User Interface Design in TADEUS, 1998,http://wwwcs.uni-paderborn.de/fachbereich/AG/szwillus/chi98ws/schlung.html

[Pinheiro da Silva, Paton 2000 #1] P. Pinheiro da Silva and N. Paton. User Interface Modelling with UML. In Proceedings of the 10th European-Japanese Conference on Information Modelling and Knowledge Representation, Saariselk a, Finland, May 2000. (To appear). http://citeseer.ist.psu.edu/article/pinheirodasilva00user.html

[Pinheiro da Silva et al. 2000] P. Pinheiro da Silva, T. Griths, and N. Paton. Generating User Interface Code in a Model Based User Interface Development Environment. In Proceedings of the International Confer16 ence on Advance Visual Interfaces (AVI2000), Palermo, Italy, May 2000. ACM Press. (To appear). http://citeseer.ist.psu.edu/pinheirodasilva00generating.html

[Gamma et al., 1994] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, ISBN-10: 0-201-63361-2,1994.

[Krasner, Pope,1988] G.E. Krasner and S.T. Pope, "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 system." Journal of Object Oriented Programming, 1988. vol. 1, no. 3, pp. 26-49, http://citeseer.ist.psu.edu/krasner88description.html

[Pinheiro da Silva, Paton 2000 #2] P. P. da Silva and N. W. Paton. UMLi: The Unified Modeling Language for Interactive Applications. In A. Evans, S. Kent, and B. Selic, editors, Proceedings of UML 2000. http://citeseer.ist.psu.edu/pinheirodasilva00umli.html

[Pinheiro da Silva, Paton 2000 #3] aulo Pinheiro da Silva and Norman W. Paton: Improving UML Support for User Interface Design: A Metric Assessment of UMLi, Proceedings of ICSE-2003 Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction, R. Kazman, L. Bass and J. Bosch (Eds.), Portland, OR, USA, 2003. IFIP, pages 76-83,http://citeseer.ist.psu.edu/pinheirodasilva03improving.html

[Pinheiro da Silva, Paton 2000 #4] Paulo Pinheiro da Silva and Norman W. Paton. User Interface Modeling in UMLi. IEEE Software, Vol.20 No. 4, July/August 2003, pages 62-69.

[Wikipedia, UML 2006] http://en.wikipedia.org/wiki/Unified_Modeling_Language, 2006

[UML ,2005 #1] Unified Modeling Language Specification,Version 1.4.2 formal/05-04-01 This specification is also available from ISO as ISO/IEC 19501,http://www.omg.org/cgi-bin/doc?formal/05-07-04

[UML, 2005 #2] Introduction to OMG's Unified Modeling Language™ (UML®), http://www.omg.org/gettingstarted/what_is_uml.htm 2006

[Abrams, Helms 2004] Marc Abrams,James Helms:User Interface Markup Language (UIML), Working Draft 3.1, 11 March 2004,http://www.oasis-open.org/committees/documents.php?wg_abbrev=uiml

[Oasis,2006] OASIS - Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/who/ 2006

[Kleppe et al.,2003] A. Kleppe, S. Warmer, W. Bast, "MDA Explained. The Model Driven Architecture: Practice and Promise", Addison-Wesley, April 2003.

[Mellor et al. 2004] S.J.Mellor, K.Scott, A.Uhl, D.Weise, "MDA Distilled: Principles of Model-Driven Architecture", Addison Wesley,March 2004

[OMG,2006] The Obect Management Group, http://www.omg.org/gettingstarted/gettingstartedindex.htm

[Murkerji,Miller 2003] Jishnu Mukerji, Joaquin Miller,OMG, "MDA Guide Version 1.0.1", omg/2003-06-01, June 2003,http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf

[Greenfield et al., 2004] J. Greenfield and K. Short. Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools. J.Wiley and Sons Ltd., 2004.

[Bråthen 2005] Erik Bråthen: A practical evaluation of software factories and the Microsoft Domain Specific Language approach for developing Web Services, Master Thesis University of Oslo, Department of Informatics 2005

[Greenfield 2004] Jack Greenfield, Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Microsoft 2004.http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/softfact3.asp

[MS Soft. fac., 2006] http://msdn.microsoft.com/architecture/factories/default.aspx

[UIML, 2006] www.uiml.org 2006

[Azevedo et al. 2000] Azevedo, P., Merrick, R. and Roberts, D. OVID to AUIML - User-Oriented Interface Modelling, in Proceedings of Towards a UML Profile for Interactive System development (TUPIS'00) Workshop, http://math.uma.pt/tupis00 York - UK.

[Limbourg et al., 2004] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon,Murielle Florins, and Daniela Trevisan “USIXML: A User Interface Description Language for Context-Sensitive User Interfaces” Workshop organised at Advanced Visual Interfaces 2004, 25 May 2004.

[comet, 2006] http://modelbased.net/comet/index.html, 2006

[Wikipedia Architecture, 2006] Architecture, http://en.wikipedia.org/wiki/Architecture 2006

[Wikipedia, Yacc 2006] Yacc, http://en.wikipedia.org/wiki/Yacc 2006

[Wikipedia, Lex 2006] Lex, http://en.wikipedia.org/wiki/Lex_programming_tool 2006

[Wikipedia UMTS, 2006] UMTS - Universal Mobile Telecommunications System, http://en.wikipedia.org/wiki/UMTS, 2006

[Select, 2006] Select Enterprise - http://www.selectbs.com/products/select-enterprise.htm, 2006

[Wikipedia Rational, 2006] Rational Rose, http://en.wikipedia.org/wiki/Rational_Software, 2006

[Sun, 2006] Sun Microsystems.com, http://www.sun.com/ 2006

Appendix E

INDEX

abstraction 3; 13; 14; 17; 20; 27; 38; 44; 53; 95; 96; 97
ActiveX 33; 35; 37
Application layer 16
application logic 26; 27; 33; 34; 36; 37; 38; 39; 48; 49; 50; 59; 74; 75; 76; 79; 90; 91; 93
application Window 67
Application-frameworks... 14
architecture .11; 16; 17; 24; 25; 27; 29; 32; 33; 34; 37; 38; 40; 42; 47; 48; 50; 52; 54; 56; 76; 87; 92; 95; 113; 116
Architecture-centric 48
business case.. 92
Business layer.. 16
business logic 34
Button 11; 66; 67; 68; 106
Button block 11; 66; 68
C++.. 32; 34; 45; 51; 71
Canvas 66
car entity 20
Car Rental System 9; 11; 19; 21; 22; 24; 101
Case study... 7; 19
Choice.. 61
client logic 33; 50; 91
CML 8; 9; 11; 15; 41; 43; 49; 52; 55; 56; 89; 93; 98; 101; 102; 109
Code generate 25; 33; 37; 39; 89
code generation........9; 15; 18; 27; 33; 39; 41; 43; 47; 55; 56; 71; 73; 74; 79; 87; 90; 97; 102
Code generation.. 7; 9; 25; 74; 85; 89
code genration 33
code skeletons.. 32
code-generation 15; 104
CodeWarrior 14
COMDEF 3; 7; 8; 9; 11; 15; 16; 17; 18; 19; 21; 24; 25; 26; 27; 28; 32; 33; 34; 36; 37; 38; 39; 41; 42; 43; 46; 47; 48; 49; 50; 52; 53; 54; 55; 56; 57; 59; 61; 65; 73; 76; 77; 81; 85; 87; 89; 90; 91; 92; 93; 94; 97; 98; 101; 102; 104; 113
Component 15; 41; 43; 66
componential approach.. 35
composite tasks.. 61
Computational Independent Model
CIM 95
conceptual Multidev dialog metamodel 63
Concurrent.. 61
Conditional 61
Constraints... 53
CORBA 15; 42; 43; 76; 87; 92; 114
customer entity 21
customer service 21; 81
Data Item 11; 66; 68; 84
Data layer 16

DCP 17
declarative models 27; 34; 36; 37
developer control 26; 33; 37
Developer control 25; 33; 37; 39; 89
development .. 3; 7; 11; 13; 15; 16; 17; 18; 25; 26; 29; 34; 35; 36; 37; 38; 40; 41; 43; 46; 47; 48; 49; 52; 53; 57; 74; 90; 91; 92; 93; 94; 95; 96; 97; 98; 99; 101; 104; 113; 114; 116
development cycle 35; 36
development environments 35
Dialog 8; 9; 11; 18; 53; 54; 55; 56; 63; 64; 65; 66; 71; 73; 82; 90; 98
dialog metamodel 63
dialog model 11; 29; 31; 32; 33; 34; 36; 62; 63; 64; 83; 84; 85; 98
distributed architectures 15
distributed environment 3; 13; 15; 17; 28; 47; 91; 99
distributed information systems
  DIS 13; 25; 28; 41
document windows 65; 67; 69
domain model 11; 34; 35; 36; 49; 59; 60; 61; 71; 72; 73; 82; 83; 90; 91; 93
Domain Specific Languages 43; 94; 96; 97; 98
DSL 3; 17; 94; 97; 98
emitter
  emitters 8; 11; 49; 56; 57; 73; 82; 85; 93; 98; 102
emitters 15; 41; 42; 98
entities 8; 16; 20; 24; 32; 36; 42; 43; 50; 59; 74; 92
environment... 3; 9; 13; 14; 17; 25; 26; 29; 33; 34; 37; 38; 39; 52; 61; 64; 71; 73; 79; 89; 95; 96; 97; 98; 113; 115
event handlers 33
events
  event 20; 21; 34; 38; 50; 51; 52; 59; 73; 75; 77; 78; 79
Evolution 7; 8; 9; 25; 28; 34; 38; 40; 91
Extensible Development Environment 97
extensible markup language
  xml 43
extension mechanisms 53
Extensions 22; 23
formal model 36
Free data items 67
Genera 7; 29; 32; 63; 114
General 7; 11; 24; 25; 26; 27; 28; 65; 66; 89; 90; 91
general block 65; 66; 68; 69; 75
generic code 25; 33; 37; 39; 89; 104
Genova 7; 8; 11; 29; 30; 31; 32; 33; 34; 38; 63; 81
Goal 22; 23
Graphic editable 25; 33; 37; 39; 89
graphical modeling tool 73
graphical notation 25; 26; 33; 37; 39; 41; 43; 89
hierarchical decomposition 36
high-level tools 13; 14; 15
Image box 66
Independency 7; 25; 26; 89
interface builders 14
Interface builders 8; 14; 38
Interface Description Language 42; 55; 116
interface guidelines 25; 26; 33; 37; 39; 89; 90
Interoperability 95

iterative development process 34
java 14; 17; 28; 32; 33; 37; 38; 45; 49; 51; 52; 56; 57; 71; 73; 74; 82; 85; 91; 92; 93; 102; 113
Java 8; 11; 14; 34; 35; 43; 45; 57; 76; 87; 95; 103; 114
Label 66; 105; 106
Language based tools 14
Layout algorithm 9; 75
leaf nodes 75
lexical language 15
lexical modeling 41
lexical models 26
Low level tools 14
Lynx 26
Macintosh 14
MBUIDE 61
MDA 3; 9; 11; 17; 43; 94; 95; 96; 97; 98; 116
metamodel 8; 9; 11; 15; 16; 27; 34; 37; 39; 41; 44; 52; 53; 54; 63; 64; 65; 67; 70; 71; 72; 76; 89; 90; 94
methodology 14; 15; 21; 29; 35; 36; 41; 43; 48; 49; 61; 76; 97; 101
Microsoft Windows 14
middleware 43; 92
Mobi-d 8; 11; 29; 34; 35; 36; 37; 38
Model based Framework 34; 39
model environment 33
Model View Controller 76
Model-based automatic generation 15; 17
model-based framework 3; 15; 17; 27
model-based strategy 17
Model-driven 95
Modeling concepts 7; 8; 25; 26; 34; 37; 39
Multi platform support 8; 9; 25; 28; 34; 37; 38; 39; 40; 91
Multidev 3; 8; 9; 11; 17; 18; 25; 41; 45; 46; 47; 48; 49; 50; 52; 53; 54; 55; 56; 57; 59; 60; 61; 63; 64; 65; 66; 70; 71; 72; 73; 74; 76; 77; 79; 81; 82; 83; 84; 88; 89; 90; 91; 92; 93; 94; 97; 98; 99; 101; 109
Multiple platform support 7; 27; 28
network 13
object model 29; 92
object selection 11; 29; 30; 31; 32; 33; 34; 63; 81
Oboe 41; 48; 114
OCL constraints 53; 54
OMG 3; 9; 17; 26; 42; 43; 45; 52; 94; 97; 114; 115; 116
Optional 61
Order independent 61
Platform Independent 43; 96; 97
platform independent models 94
Platform Specific Models 43; 96
Portability 94
presentation model ...9; 11; 36; 49; 60; 61; 62; 71; 72; 73; 75; 76; 77; 78; 79; 82; 83; 84; 85; 89; 90
Primary actor 22; 23
primitive tasks 61
Productivity 7; 9; 24; 25; 33; 36; 39; 89; 94
programming language 38; 39; 45; 76
renderer 11; 50; 51; 73; 77; 78; 85; 86; 87; 89; 91

rental service 20; 21; 81; 102
Repeatable 61
reservation entity 20; 21
Sequential 61
services 8; 17; 20; 21; 24; 32; 42; 43; 50; 59; 61; 74; 81; 85; 87; 89; 92; 93; 102
shortcomings 9; 24; 89; 91; 92; 93; 99
SINTEF 41; 52; 92
skeleton task model 35
Software engineering 15
Software Factories 9; 96; 97; 98; 116
Software Factory Schema 96; 97; 98
Software Factory Template 97; 98
source code 13; 19; 32; 35; 37; 38; 39; 41; 42; 47; 73; 92; 95; 96
Standard Generalized Markup Language
  sgml 43
state object 61; 62; 71; 73; 74; 77
stereotypes 15; 16; 32; 41; 52; 53; 54; 55; 63; 64; 67; 82; 89
Stereotypes 53
Sun Microsystems 45; 113
SUN Microsystems 14
System evolution 24
Tabbed pane 66; 69
Tagged value 54
target platform 28; 33; 45; 51; 96
task model 11; 35; 36; 49; 60; 61; 62; 71; 72; 73; 76; 77; 79; 84; 85; 90; 91; 93
Template support 25; 33; 39; 89
templates 33; 37; 38; 39; 90; 97
test case 19
three-tier model 32
toolkit 14; 71
transformation 44; 94; 96; 98
Tree view 66; 69
UIML 8; 9; 11; 44; 45; 49; 50; 51; 75; 76; 79; 85; 86; 89; 90; 91; 93; 98; 105; 114; 116
UIML mapping 9; 75
UML 3; 8; 9; 15; 16; 17; 19; 26; 27; 29; 32; 33; 34; 37; 39; 41; 43; 44; 45; 47; 48; 49; 52; 53; 54; 55; 56; 57; 63; 64; 65; 81; 82; 88; 89; 90; 92; 93; 97; 98; 101; 114; 115; 116
use case 21; 22; 23; 29; 32; 36; 61; 74
use case driven 21; 29; 61
Use case driven 48
use cases 11; 21; 22; 29; 32; 36; 49; 61; 74; 101
User Interface Markup Language 44; 116
user interface parts 26
user interfaces 3; 7; 13; 14; 15; 17; 24; 25; 26; 27; 29; 30; 33; 35; 36; 38; 44; 45; 47; 49; 50; 56; 61; 63; 64; 73; 85; 89; 90; 91; 92; 93; 97; 98; 113; 115
user task model 35
userservice 16; 17; 19; 20; 24; 32; 38; 42; 50; 81; 82; 87; 92; 93; 109
viewpoint 13; 21; 42; 95; 96; 97; 101
Visual Basic 32; 34
Visual Café 8; 29; 38; 39; 40
widget 14
work protocol 8; 29; 35; 38
World Wide Web consortium
  W3C 43
XMI 8; 26; 43; 49; 52; 89; 93; 114

XPath 44
XSLT 8; 44; 49; 52; 114
XSLT stylesheet 44