wordt
NIET
uitgeleend

MOBILE DEVICES AND ELECTRONIC-CRM

Graduation Thesis

Binne Simonides

RuG

Atos Origin

MOBILE DEVICES AND ELECTRONIC-CRM

AUTHOR(S) : Binne Simonides
VERSION : Final
INSTITUTION : Department of Mathematics and Computing Science, University of Groningen
CUSTOMER : Atos Origin TUM
SUPERVISORS : Ir. R.K. Rabbers (Atos Origin TUM)
: Dr. R. Smedinga (RuG, dept. Mathematics and Computing Science)
: Dr. Ir. T.D. Meijler (RuG, dept. Management and Organization)
DOCUMENT DATE : 4-07-2007
NUMBER OF PAGES : 101

Abstract

Electronic Customer Relationship Management (eCRM) is a large enterprise application domain with complex user interaction. The dynamic market of Mobile devices introduces a lot of variability with respect to capabilities like screen-size, operating system, network support and many others. The combination of these two areas into one application makes maintainability both very important and difficult.

In this thesis the primary focus will be on the possibility of increasing the maintainability of such an application through model-driven development. The following issues and solutions will be addressed. The possibility of refining a software design through a Model Driven Architecture (MDA) approach in which high-level business models are mapped to platform specific models in a multi-tiered architecture. The applicability in a model-driven development of decoupling mechanisms used by design patterns like Model View Controller (MVC) or Presentation Abstraction Control (PAC) and the possibility of using a run-time thin client approach called Wireless Universal Resource File (WURFL) for supporting the differences between the many different mobile devices.
Finally this thesis introduces a new form of model-driven User Interface development called Tailorable Presentation Control Classes (TPCCs). TPCCs find an optimum between User Interface development technologies in which a User Interface directly manipulates the application objects but are not model-driven and technologies that do support model-driven User Interface development but require complex powerful models.

Contents

1 Introduction

The assignment for this thesis has been established through a joint venture of Atos Origin Nederland B.V. and the University of Groningen. In order to graduate for my Computing Science study at the University of Groningen, I started with my graduation assignment in July 2006. Atos Origin TUM provided me the opportunity to graduate on a very interesting subject. At Atos Origin I was supervised and supported by Roelof Rabbers and at the University of Groningen I found Rein Smedinga and Theo Dirk Meijler willing to supervise me.
In this chapter a background description of Atos Origin TUM, the University of Groningen (RuG), the research question and the structuring of this thesis will be explained.

1.1 Atos Origin TUM

Atos Origin TUM develops Customer Relationship Management (CRM) applications for large companies that provide services in the field of Telecom, Utilities and Media. Currently, their most important product is the OrderManager, an ordering system for making agreements between customer and supplier about the delivery of products or services.
The Solution and Design (S&D) team of Atos Origin TUM would like to add a Sales Support System (S3) to their software suit. In this way not only the ordering, but also the sales process will be supported.
The sales process exists of bringing together supply and demand, in other words, coming across with a good solution or the right product.

The Sales Support System has several sales channels, including:

- PC/Internet
- TV/Internet
- Mobile Phone/Internet
- PDA/Internet

The focus of this thesis project will be the sales and support through mobile phones.

Business vision:
The Sales Support System is developed to assist in the creation of an all digital lifestyle. This means that customers can order products and services with minimal human intervention. Human activities are a major expense; any reduction in this field is very lucrative.

Business rationale
The addition of the mobile phone channel to the Sales Support System delivers a lot of extra possible services that can be offered to a large group of customers.

1.2 University of Groningen, faculty of Management and Organization, Dept. of Information Systems

The department of Information systems from the faculty of Management and Organization within the University of Groningen (RuG) is doing research on the subject of how IT can support business flexibility in the context of large enterprise applications like CRM. A Business-Model-driven development Approach (abbreviation: BMA) is a Model-driven approach that supports this business flexibility and is based on business modeling, in which from a high level business model IT elements can be generated.
The MDA (Model driven Architecture) is a widely adopted model-driven development approach that is known to enable code generation from UML-based (software) models. The MDA enables the possibility to map to different underlying execution platforms. Thus, model-driven development approaches enable flexible model to platform bindings.

The distinguishing features of a business-model-driven development approach with the MDA and many other Model-driven development approaches is that the business models are the basis of the development, not software models. Thus, in a BMA, elements of the business model, such as the activities, the business objects,

the organizational structures are all mapped to software based solutions. Moreover, in a BMA fixed aspects of the application architecture, such as e.g. a multi-tiered Service-Oriented Architecture, are not modeled. Thus, the BMA can also be seen as a Variability Management approach: Business-models describe what is variable in Enterprise Systems.

Even where business-model-driven approaches have been tried out in some degrees, there are various issues that need further research:

1. The interaction between business models and software models: There is a good chance that not all software behavior –especially component implementations– can be described using business models. In that case extra software implementation modeling may be needed. The question is than what is precisely the relation between such software models and business models, given that now business models are no longer just requirement models. This relation is similar to the Platform Independent Model (PIM) → Platform Specific Model (PSM) in MDA.

2. Maintaining business applications with complex user interactions. Business applications with complex user interactions, e.g. where the user manipulates object structures, such as in electronic-CRM (eCRM) have typically been developed in tight-coupled technology, as server based or fat client based applications. Such architectures no longer fit-in the Service-Oriented Architecture with its standard Simple Object Access Protocol (SOAP) message-based decoupling between UI and services. For such applications, (SOAP) message based (technical) decoupling is principally possible, given the improved bandwidth of infrastructures, but this will typically still lead to a strong functional coupling between the UI and the server component. The problem of such functional coupling (as of any form of coupling) is that it hampers maintainability. A (business) model-driven approach can play an important role in solving this maintainability problem, since the generation of both the server component and the client UI can be based on one (business-) object model.

This thesis assignment will constitute a research of the issues mentioned above. These two issues are directly related to large enterprise applications in the domain of CRM which are developed by Atos Origin TUM.
The remainder of this chapter is organized as follows; first a context description of CRM, mobile devices and a geographical picture of the mobile environment will be given. Secondly the research question is formulated, and finally the thesis structure will be explained.

1.3 CRM, eCRM and eTOM

CRM: (Customer relationship management) is a broad term that covers concepts used by companies to manage their relationships with customers, including the capture, storage and analysis of customer information [2]. The different aspects of CRM are graphically illustrated in *Figure 1.*

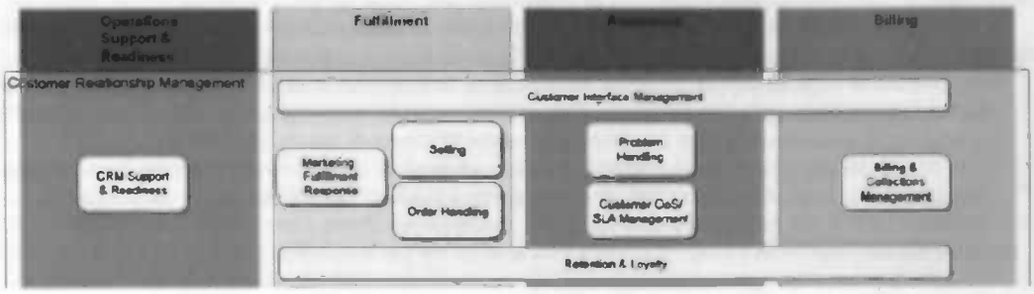

Figure 1 CRM

eCRM: is the electronic based version of CRM. eCRM uses the internet to increase the relationship to the customer. eCRM itself is a very *large application* which is part of an even larger application called eTOM.

A phrase that captures the essence of eCRM is:

"eCRM is all about customers choosing a product."

eTOM: (enhanced Telecom Operations Map) is a guidebook, the most widely used and accepted standard for business processes in the telecommunications industry. The eTOM describes the full scope of business processes required by a service provider and defines key elements and how they interact. eTOM can be compared to ITIL which is an analogous standard or framework for best practices in information technology [3].
The different aspects of eTOM are graphically illustrated in Figure 2, in the right corner of Figure 2 CRM and its relation to eTOM is illustrated. For a more complete and detailed description of the different aspects of CRM and eTOM which are mentioned in the Figure 1 and 2 one can refer to [4]. For now it is important to notice that eTOM, CRM and eCRM are large applications, in section 3.1 there is a further analysis of large applications.

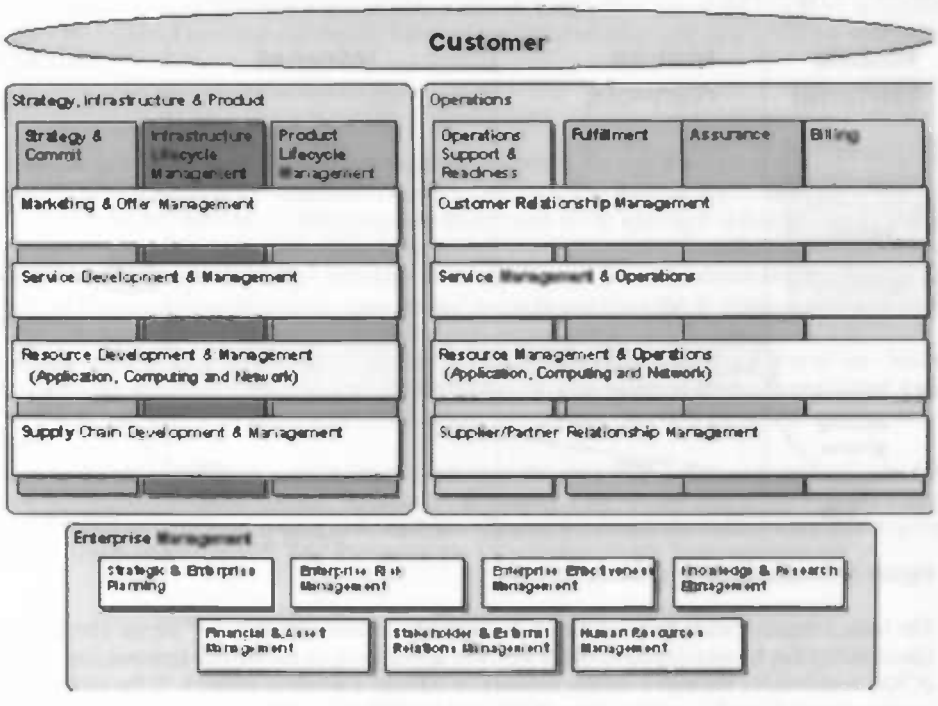

Figure 2 eTOM

1.4 Mobile devices

At the present time there are about 400 different mobile devices in the European market. It is expected that by 2008 there will be over 500 different handset models in large markets, such as Western Europe [12].

Mobile devices have about 100 different capabilities [6]. These capabilities vary from Short Message Service (SMS) which is available on basically every mobile handset, to J2ME[1] and sophisticated native applications which will be available to a more limited subset of the available handsets.
Another important technology that is directly related to the mobile device is the Mobile Network; properties like latency and available bandwidth have an influence on the application.

Besides the variability in functional capabilities handsets also have many features in common with one another. Many devices coming from the same manufacturer are often an evolution of the same hardware and software. For example the differences between the Nokia 7110 and a Nokia 6210 are minimal. Also devices from different manufacturers may run the same software. For example both Eriksson and Nokia use the Symbian Operating System and Openwave Systems provides the browser for Siemens, Motorola and Samsung.
The commonalities and differences between the different devices can have an important impact on the application, for example variability in screen size could have an important effect on the user interface.

1.5 Geographical environment

The application has to be deployed into the real-world the picture below sketches the geographical environment for mobile applications.

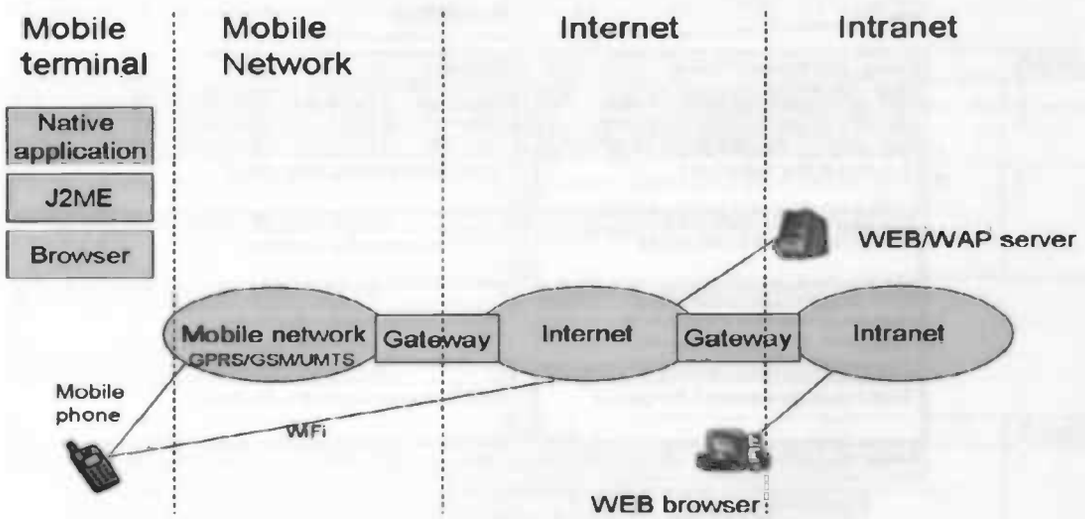

Figure 3 Geographical environment

The basic communication between a mobile terminal and some WEB/WAP[2] server which hosts the eCRM functionality can be established in two ways. The application on the mobile terminal can communicate with the eCRM functionality through a mobile network or through a wireless network. If the mobile terminal communicates through a mobile network like GSM/GPRS the communication requires a Gateway which can convert the requests to and from the eCRM functionality in a suitable format for the mobile terminal. The communication can also take place through a wireless network like WiFi; this kind of communication is

[1] J2ME (Java 2 Platform, Micro Edition) is a set of Java API's which is specifically developed for resource constrained devices like mobile phones.
[2] WAP (Wireless Application Protocol) WAP sites are mobile internet sites which allow for interactive communication with a mobile device.

available for the more expensive mobile phones and does not require a special gateway for translation to a specific format.

1.6 Research question

In the beginning of the internship an analysis has been done about eCRM and mobile devices; this analysis led to the following main research question.

Which generic IT-Architecture is suitable for using eCRM functionality through mobile devices?

The initial research shows that the application should be able to handle the variability (functional flexibility) with respect to the large functional domain of eCRM and the large variability in the field of mobile device capabilities. The combination of these two domains into one application will constitute a complex architecture (infrastructure) with a lot of dependencies. For handling the required functional flexibility and decouple it from non-functional device capability issues it has been chosen to use Model Driven Architecture (MDA) in combination with a Service Oriented Architecture (SOA). MDA allows for handling functional flexibility and non-funcitonal capabilities at different levels of abstraction at design time, the usage of a SOA is expected to be able to supply the required run-time decoupling between the different platforms that fulfill the application.

1.7 Thesis structure

Chapter 2 will consist out of an analysis of the requirements which the application has to fulfill. Due to the large size of the total domain of the application it was found usefull to formulate a focus and that the research is going to be done from an "approach". The focus and description of the approach is described in Chapter 3.
Chapter 4 contains the made design decisions, these design decisions present a number of high level hypothesis for solving the requirements. The design decisions are made through a more in-depth analysis and literature investigation of the available architectural concepts and technologies from the developed approach. When an analysis does not result in a design decision a challenge is formulated.
Chapter 5 consists out of an enumeration of the expected challenges from chapter 4. Some of the challenges are subjected to a further analysis before a design decision is taken. The design decisions from chapter 4 and chapter 5 together will lead to an elaborated model which will present "the best guess" for solving the requirements.
The Proof of Concept from chapter 6 splits the elaborated model into three parts and investigates the first part by an actual development of the proposed "approach". In chapter 7 the second part of the Proof of Concept is investigated through an additional source code analysis. Chapter 8 contains the Future work and chapter 9 formulates the conclusion.

2 Requirements

In the previous chapter eCRM and mobile devices are described and the research question: "*which generic IT-Architecture is suitable for using eCRM functionality through mobile devices*?" is formulated. This research question is general and the domain of eCRM and mobile devices is large. Together with Atos Origin TUM and the University of Groningen a list of requirements has been developed. This list of requirements is intended to keep the scope of the problem-domain clear and manageable.

The requirements are prioritized, where priority 5 is the highest property and will be the main focus of this thesis. Due to the size of the problem domain the requirements with a lower priority are moved to the Future work (see chapter 8).

Throughout the remainder of this thesis references to a specific requirement will be indicated by a [NFR_id] notation, where for example [NFR02] indicates a relation or conflict with non-functional requirement NFR02. A special paragraph business requirements has been devised, the reason for this is that these specific non-functional requirements have an important but indirect effect on the functional and non-functional requirements.

Functional requirements:

ID	Title	Description	Priority
FR01	User interface	A readable UI has to be presented to the user	5
FR02	Stateful/Stateless transactions	The application which is presented to the user, manipulates objects through Stateful transactions	5
FR03	Dialogue handling	The application should be able to handle different dialogs from different devices properties like WAP, J2ME, screen size and cashing in a consistent manner.	5

Non-functional requirements:

ID	Title	Description	Priority
NFR01	Unreliable connection	The application has to take into account the unreliable connection i.e. intermittent connectivity properties of the mobile network.	3
NFR02	Device Flexibility	The application should run on as many phones as possible, i.e. scalable with respect to the addition of new mobiles.	5
NFR03	Functional Flexibility	The application should be able to handle new eCRM functionality easily, i.e. extendible with new eCRM functionality. New eCRM functions should be easy to compose.	5
NFR04	Bandwidth dependable appearance	The appearance of the application depends on the available bandwidth.	5
NFR07	Long running transaction reliability	It might be that some eCRM transactions are long-running them selves. The unreliable nature of the mobile connection i.e. long off time can also require the need for long running reliability i.e. partly filled in forms.	1
NFR08	Interoperability	Because the eCRM application will be large it is likely that the application will be deployed on	4

		different systems which all have to work on the same data. This in combination with the fact that there are a lot of different mobile vendors (Nokia, Siemens and Eriksson) with all their own system implies that there will be Interoperability issues.	
NFR09	Performance	The outlook of the application should have minimal/no effect on the performance. The performance of the application should be acceptable for the customer.	4
NFR10	Usability	The UI must allow for easy and intuitive navigation.	4
NFR11	Device switching	The application can be shared between different devices (pc, TV, phone), this should not leave the eCRM application in an inconsistent state. For example if partially filled-in forms with the mobile are finished with the pc. "This requirement has overlap with [NFR07]"	1
NFR12	Legacy systems	The application might (not intended) interact with legacy systems.	1

Non-functional business requirements:

ID	Title	Description	Priority
NFR05	Functionality type reliability	Different eCRM functions impose different risks for the application i.e. a product-order has much more risks than a product-view. eCRM functionality should be categorized.	1
NFR06	eCRM QoS metrics	The properties of the eCRM functionality provided by the business logic should be able to be categorized in QoS measurements. Some eCRM transactions have a higher impact on the relation with the customer than others. If something goes wrong in a product order transaction, the impact is much higher then if a customer views a list of products and something goes wrong. This kind of measurement can probably be visualized in graphs, with for example risk on the x-axis and customer impact on the y-axis.	1

2.1 Requirement analysis

2.1.1 Unreliable connection [NFR01]

The unreliable connection properties of the mobile connection like intermittent connectivity, zero connectivity (incomplete network coverage) can result in a long offline-time of the mobile devices.

2.1.2 Device flexibility [NFR02]

The eCRM functionality has to be available for "all" mobile devices. Solutions for this kind of flexibility in for the mobile phone strongly depend on the available functionality on the mobile device. As is mentioned in *section* 1.4 mobile devices have a lot of commonalities but also a lot of variability in available functionality. Another important factor that also has to be taken into account is the demands on the available eCRM functionality itself. Specific "difficult" eCRM functionality might require a sophisticated native or J2ME solution, while other relative "simple" eCRM functionality might have a simple browser solution.

2.1.3 Functional flexibility [NFR03]

The functional flexibility consists out of the easy expansion of existing eCRM functionality with new eCRM functionality and easy composition of functionality into new functions i.e. existing functionality should be able to be reused.
The functional flexibility also consists out of a consistent manner for implementing the new eCRM functionality on the mobile device.
It might be that [NFR02] and [NFR03] are conflicting because for functional flexibility you don't want to implement all new functionality differently for every device. On the other hand, implementing functionality differently on every device might be required to be able to deliver a lot of different functionality.

2.1.4 Bandwidth dependable appearance [NFR04]

The different available mobile networks like UMTS, GPRS and GSM vary in offered bandwidth. The application should be capable of representing a different outlook based on these differences in bandwidth. For example UMTS has a bandwidth of 384 kbit/s; this might allow the application to have a nicer look by including pictures.

2.1.5 Functionality type reliability [NFR05]

The eCRM application will be large; this means that there will be a lot of different types of functionality. These different types of functionality can differ from simple functionality such as viewing a product description to more complicated functionality such as ordering a product.
The different types of functionality can pose different risks on the eCRM application. If for example a customer order fails and the customer gets a double order it is very likely that the customer will not use the application again, while a failure of a product view will probably have less negative effects. A categorization of eCRM functionality can give use full information with respect to specific demands on the implementation. For example a simple function with a high risk can indicate that the function becomes complicated i.e. it needs a additional code to reduce the risk.

This requirement has been moved to Future work chapter 8. In this section some categories to judge the eCRM functionality are proposed.

2.1.6 eCRM QoS metrics [NFR06]

This requirement is strongly related to [NFR05]. The purpose of this requirement is to sketch the QoS of eCRM functions against for example relevant geographical properties of the application. For example it could be that the reliability of the UMTS network is higher than that of the GPRS network.

This requirement has been moved to Future work chapter 8. In this section some categories to judge the eCRM QoS metrics are proposed.

2.1.7 Long running transactions [NFR07]

It might be that some eCRM transactions are long-running themselves, maybe this kind of metric could be placed in [NFR05].
The unreliable nature of the mobile connection resulting in long off-line time can also require the need for long running reliability i.e. partly filled in forms.

2.1.8 Interoperability [NFR08]

Due to the size of the domain of the total application it can be expected that there will be a lot of different systems which all have to work on the same data set. These kinds of distributed systems will usually suffer from Interoperability issues.

2.1.9 Performance [NFR09]

The performance of the application i.e. the response time of the application on the mobile should be acceptable. The latency on mobile networks and the size of the eCRM application are important factors that will have an influence on the performance of the application.
The outlook of the application is less important than the performance of the application. This requirement has a relation with [NFR04].

This requirement has been moved to Future work chapter 8. In this section some measures are discussed to increase the performance of the application on the mobile client.

2.1.10 Usability [NFR10]

The UI on the mobile phone should allow for easy and intuitive navigation by the customer. The small screen and the limited keypad of the mobile phone make this a difficult requirement.
The small screen can only display a limited amount of information compared to a desktop computer and the keypad of the mobile only allows for a limited set of navigational opportunities.

2.1.11 Device switching [NFR11]

The application should be able to switch between the different channels of the Sales Support System (S3) i.e. the phone, television and pc. It should for example be possible to fill in parts of some eCRM function on the phone and finish it with the mobile. This kind of switching should not leave the application in an inconsistent state. This requirement has a relation to [NFR07] i.e. it will be long running.

This requirement is not handled.

2.1.12 Legacy systems [NFR12]

It is possible that the system might interact with legacy systems in the future. An analysis at this stage does not make a lot of sense, but in the future it might occur that the application has some benefit of accessing some legacy system.

This requirement is not handled.

3 Initial Model

Due to the complexity of the total domain of the application it is decided that it would be useful to formulate a focus and that the application is going to be developed from a specific approach. This chapter is organized as follows after the formulation of a focus the high-level concepts from the approach are explained. Next an initial research of the problem domain is conducted; this initial research will investigate the eCRM functionality, the variability in the field of the mobile device capabilities and a Model driven development technique called Model Driven Architecture (MDA). The initial research will lead to a more concrete formulation of the high-level concepts from the approach.

Focus:
The focus for this thesis will be on the *maintainability* of the application.

Definition Maintainability:
The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment [16].

It is expected that large applications like eCRM, high variability in mobile devices in combination with the (non) functional requirements like device/functional flexibility will have a large impact on the maintainability of the application. This expectation which is explained in the remainder of this chapter is the main reason for choosing maintainability as the key focus for this thesis.

3.1 Large applications

In section 1.3 a description of the eCRM functionality is given. The important fact of this section is that eCRM is a large and complex enterprise application which will consist out of lots of components and is influenced by a lot of different forces. Forces that have to be considered in large applications are for example [17]:

- Localizing changes to one part of the solution minimizes the impact on other parts, reduces the work involved in debugging and fixing bugs, eases application maintenance, and enhances overall application flexibility.
- Separation of concerns among components (for example, separating the user interface from the business logic, and separating the business logic from the database) increases flexibility, maintainability, and scalability.
- Components should be reusable by multiple applications.
- Independent teams should be able to work on parts of the solution with minimal dependencies on other teams and should be able to develop against well-defined interfaces.
- Unrelated components should be loosely coupled.
- Various components of the solution are independently deployed, maintained, and updated, on different time schedules.
- Crossing too many component boundaries has an adverse effect on performance.
- To make a Web application both secure and accessible, you need to distribute the application over multiple physical tiers. This enables you to secure portions of the application behind the firewall and make other components accessible from the Internet.

To incorporate quality attributes like maintainability, reusability, scalability, reliability and security in complex enterprise applications like eCRM; following [17] and many others a layered application is proposed. What kind of layering will further detailed below.

Maintainability issue example:
It is general knowledge that all large applications suffer from maintainability. The average effective productivity of for example a programmer on windows OS results in about 1 line of code per month, this implies that for every 100 bug-fixes they get 99 new bugs back.

3.2 Approach

The basic approach is intended to increase the maintainability of the application by splitting the total application into manageable pieces. For developing eCRM functionality on mobile devices the application is divided into three main areas; static platform(s), runtime and design time.

Static part (platform):
The static part of the architecture consists out of the part that is fixed and does not change. On a high level the static part of the architecture consists out of a Tiered architecture and its main components (see Figure 4). On a lower level the static part consists out of underlying execution platforms such as a multi-tiered Service Oriented Architecture or a Web application platform. The initial research on the eCRM functionality will lead to a more concrete formulation of the specific tiered architecture.

Runtime part:
The runtime part of the architecture is the part that changes dynamically at runtime. In our case this is the variability in the hard/software capabilities of the mobile devices and available networks. This variability can only be detected at run-time but might be modeled at design-time.

Design time part:
The design time part of the architecture is the part that changes at design time. In our case this is the variability in the eCRM functionality.

The structure of the remainder of this chapter is based on this static/runtime/design time subdivision.

3.3 Layered applications – static

The maintainability of a large application is positively influenced if components can be easily changed and removed. Within a layered architecture layers usually only communicate with the layer above and below, this kind of decoupling within an application through layers increases the low coupling and high cohesion of the architecture. The high cohesion and low coupling properties between layers increases the ability of components within the different layers to be modified replaced and developed independently of each other.

Some obvious candidates for a large application with a strong graphical user interface component are; presentation layer, application (business logic) and a data layer. A layered architecture which is split up in these three layers is called a 3-Tier architecture. This constitutes the static part of the approach.

Presentation layer:
A separate presentation layer increases the maintainability of the application because it separates the User Interface issues of the different mobile phones from the implementation of the core eCRM functionality [NFR03].

Application layer (business logic):
The application layer is the obvious candidate for the implementation of the business logic of the eCRM functionality. The use of an application layer results in an increased abstraction which allows for the decomposition of eCRM functionality into manageable pieces [NFR03].
Layering also increases the reuse of existing components which has a positive effect on reuse of existing eCRM functionality [NFR03].

The application layer itself can be split up into additional layers itself this kind of additional separation can be used to decompose the eCRM functionality even more. A separation of the application layer into additional layers transforms a 3-Tier architecture into an n-Tier architecture. A known use for a fourth tier is for example some sort of data pre-formatting functionality that is required by the (core) business logic.

Data layer:
A separate data layer allows the application for a separating the permanent storage issues from the specific eCRM functionality.

3.3.1 3 and 4-Tier architecture

As is mentioned in the section above the application layer can be split up in multiple layers itself i.e. making it an n-Tier architecture.
The subdivision of the application layer can have positive effects on the maintainability of the application because it allows for a further decomposition of the eCRM functionality. Due to the large domain of the eCRM functionality it can be expected that a further decomposition of the eCRM functionality indeed increases the maintainability.

Another factor that suggests the use of a 4-Tier architecture is the fact that the Sales Support System (S3) application will be joint project of the RUG and Atos Origin this implies that there will be many different teams working of the project for short periods of time. The use of a 4-Tier architecture allows for a clearer separation of the total application which allows different teams to work more independently of each other.

A negative effect of 4-Tier architecture is that it increases the complexity of the application; this requires the need for software engineers to develop a thorough understanding of the architecture of the application.

In this thesis a 3-Tier architecture and not a 4-Tier architecture is taken as the basis for the approach. A 3-Tier architecture is easier to understand and a more thorough investigation of the eCRM functionality would have to be done to give this extra Tier some meaning.

Decoupling:
To further increase the flexibility of the eCRM business logic, the eCRM business logic is decoupled from the presentation logic for the mobile client through a separate interface logic component (not a separate layer as the picture might indicate) which communicates with the layers above and below. The goal of this interface component is to minimize the effect of functionality additions in the business logic on the presentation logic i.e. they can be handled separately and thereby improving the functional flexibility [NFR03]. Another positive effect of this loose coupling between presentation and business logic is the increased Interoperability [NFR08].

The picture below illustrates the 3-Tier architecture in combination with the interface logic that is going to be used as the basis for developing the static part of our approach.

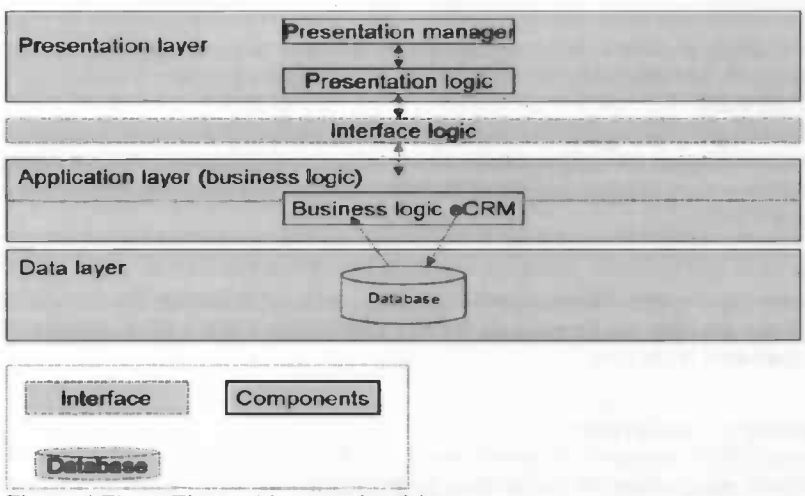

Figure 4 Three-Tier architecture (static)

3.3.2 Thin client + Thick client:

Within a layered architecture another important distinction can be made i.e. the thin and thick client approach. This distinction determines where parts of the presentation and application layer will be implemented i.e. on the client side or on the server side. The research field of thin and thick clients is quite large itself i.e. there are intermediate solutions, for a more detailed description section 4.5 can be consulted. For this initial analysis only the properties of the thin and thick client and their relation with the relevant requirements are considered.

A thin client uses only a small portion of the presentation layer i.e. presentation manager (see section 4.5) is located on the client. A thin client is client uses the resources of a host computer (server) to do the required data processing. The best known example of a thin client is the browser. A thin (mobile) client is expected to have a positive effect on [NFR02] and [NFR03].

In a (ultra) thick client the application layer (business logic) is placed in its totality on the client. It does all of its data processing himself and only depends on a server for communication data which it wants to store permanently. A thick client is expected to have a positive effect on [NFR09] and [NFR10].

3.4 Mobile devices — run-time

As described in section 1.4 it is expected that there will be a lot of variability within the capabilities of the mobile devices; this kind of variability has to be handled at run-time. The table below is a short description of the variability that will probably have to be taken into account by the application.

Software capabilities:	**Hardware capabilities:**
Native applications	Screen-size
J2ME	Resolution
Flash	Cache-size
HTML "advanced browser"	Operating system
XHTML-MP "simple browser"	Storage-size
SMS	Mobile Network support (UMTS/GPRS/GSM)
	Wireless Network support (WiFi)
	Bluetooth

It is expected that the variability in software/hardware capabilities will have an important effect on a few non-functional requirements i.e. Device Flexibility [NFR02], Functional Flexibility [NFR03], Performance [NFR09] and Usability [NFR10].
In this chapter the focus is on [NFR02] and [NFR03] because they can be directly related to the maintainability. Other requirements like [NFR09] and [NFR10] also have an impact on the maintainability but this relation is less obvious. Section 4.5 has a more detailed analysis of these requirements and their expected relation to the maintainability of the application.

Variability:
The variability that is introduced by the different mobile devices is considerable and has to be detected at run-time. There are technologies available like for example WURFL (see section 4.8) that can handle the variability in for example screen-resolution at run-time.

3.5 Model Driven Architecture – design time

MDA is a technology that is going to be used for handling the design time variability in the application, in section 4.1 there is more elaborate description of MDA. Basically MDA is an iterative top-down software engineering design approach that allows the refinement of a software system through defining models at different levels of abstraction. The different levels of abstraction allow a system to suppress irrelevant detail, this suppressing of irrelevant detail is very important in large applications to keep it maintainable [27].
At a high level the business functionality (eCRM business logic) is specified through a Platform Independent Model (PIM) without any technical detail. At the Platform Specific Level (PSM) technical detail about the application can be added.
MDA is a design time approach which can be used for generating code into the context of the 3-Tier architecture (fixed) i.e. new eCRM functionality can be modeled in one PIM and generated into the business and presentation logic of the architecture through there associated PSM's where it is runtime executed.

The main idea in this thesis is that the MDA approach can be used for the generation of code for each of the different layers through their associated PSM's from one central PIM; this keeps the PIM in sync with the variability of the application. In section 4.3 the Twin Object principle is introduced which is basically a further refinement step of this main idea.

Variability:
In the total architecture there is a lot of variability, this variability consists out the variability in the different mobile devices, available networks and eCRM functionality. The difference in mobile devices expresses its variability in screen-resolution, memory, operating system, available network. The variability in the business logic expresses itself through the available eCRM functionality. All the design time variability introduced by the application is going to be captured in models (PIM/PSM).

PIM high level business variability:
It is expected that the PIM can be used to model the high level variability in the eCRM functionality; it models the commonalities of the eCRM functionality.

PSM level variability:
It is expected that the PSM's (for the presentation layer/business logic) which are derived from the same PIM can be used to model the low level mobile device and network variability. The PSM model can be expanded with UML meta-model constructs like <<stereotypes>>, {tagged-values} and variation points to model low-level variability (see section 4.1).

4 Design decisions

In this chapter a more detailed analysis of the Initial model (see figure 4) from the previous chapter will be done; this analysis is done through a literature investigation and will lead to a refinement of the Initial model. If an analysis leads to expected problems a challenge is formulated. The challenges are collected in chapter 6 and analyzed more thoroughly before a design decision is made. The design decisions from this chapter and chapter 6 will lead to an Elaborated model (see section 5.8) which is the basis for the Proof of Concept from chapter 7.

For the further analysis we will use a top-down approach starting with high level architectural concepts like MDA, 3-Tier architecture and thin/thick client approaches. After the high level analysis we will work down and eventually end up with the enabling platform technologies like Java, jsp and servlets which will be used for implementing the Proof of Concept.

4.1 MDA

Model Driven Architecture[3] (MDA) is a Model Driven Development (MDD) approach which is used for large enterprise applications. MDA is sponsored by the Object Management Group (OMG). MDA supports model driven engineering of large enterprise applications by providing a set of guidelines for structuring specifications expressed as models [22].
MDA is an iterative top-down software engineering design approach that allows the refinement of a software system through defining models at different levels of abstraction. The different levels of abstraction allow a system to suppress irrelevant detail, this suppressing of irrelevant detail is very important in large applications to keep it maintainable [27].
One of the main aims of the MDA is to separate design from architecture. Realization technologies facilitating the design and architecture can alter independently. The design addresses the functional (use case) requirements while architecture provides the infrastructure through which non-functional requirements like scalability, reliability and performance are realized. MDA envisages that the platform independent model (PIM), which represents a conceptual design realizing the functional requirements, will survive changes in realization technologies and software architectures [27]. Below the important principles and standards which are used in MDA are described.

Refinement principle (main idea)
For this thesis the main idea is that the MDA approach can be used as a *refinement principle*. This refinement principle uses the MDA approach such that from one central PIM the code is generated for each of the different layers (platforms) through their associated PSM's. This refinement principle increases the consistency between the different layers of the application and thereby keeps the PIM in sync with the variability of the application. In section 4.3 the *multiple refinement principle* called the *Twin Object principle* is introduced; this is basically a further refinement step of this main idea.

Note: MDA also allows for a different usage than the refinement principle as used in this thesis. MDA allows for portability i.e. it is easy to replace one PSM with another PSM; this makes MDA suitable for withstanding changes in time.

Meta Object Facility
The Meta Object Facility (MOF) is a standard for model driven engineering [27]. The MOF can be used to describe/define object oriented meta-models like UML or non-object oriented meta-models like Web Services.

[3] The term architecture in MDA doesn't refer to the architecture (as defined in *IEEE 1471*) that is being developed but to the architecture of the various standards and model forms that serve as the technology basis for MDA.

The Meta Object Facility (MOF) is a core concept within MDA. It is a four layered architecture which is capable of defining meta-models. MOF is a standard which can be used to write meta-models.

Meta-model
A meta-model is an abstraction on top of a model which describes certain properties of the model. A model is an abstraction of certain entities in the real world. A typical meta-model that is endorsed by OMG is UML or CWM, but MOF allows for developing different meta-models which might be more suitable for capturing the domain specification of a specific model. If one for example envisages the world as a collection of objects, the UML meta-model formalizes object properties [27].

Model
A model can be seen as an abstraction of phenomena in the real world.
A model is a formal specification of the structure and/or function of a system.

Platform Independent Model
The system functionality can be described using a Platform Independent Model; the PIM model is independent of the technological platform that is used to implement this functionality. The PIM is used for modeling the functional requirements of the business application, eCRM functionality in our case [27].

Model Transformation Language
The PIM can be transformed into one or more Platform Specific Models (PSM's) using a Model Transformation Language (MTL). The transformation of a model (PIM) into other models (PSM's) refines the application.

Platform Specific Model
A platform-specific model (PSM) is a model of a software or business system that is linked to a specific technological platform (e.g. a specific programming language, operating system or database). The PSM can be extended with UML and OCL constructs to add detail to the application, for example specific presentation issues that are independent of the core business functionality should be handled here [27].

Stereotypes
Stereotypes are indicated through a <<stereotype>> construct. Stereotypes are UML-constructs that can be applied at the PSM model for adding specific information required at that specific PSM level. For example; at the presentation layer PSM a <<button>> stereotype indicates the presence of a button UI element.

Object Constraint Language (ocl)
The Object Constraint Language (OCL) is a formal language which is used to express constraints and preconditions. These constraints indicated by '{}' brackets specify conditions for the system that is modeled [35]. OCL-statements can be used to express application specific constraints on the PIM and PSM levels.

Source Code
The source code which implements the application can be generated from the PSM's and deployed on for example a server. The generated Jsp's or Java code for example can be deployed into a servlet container or application server like tomcat or glassfish.

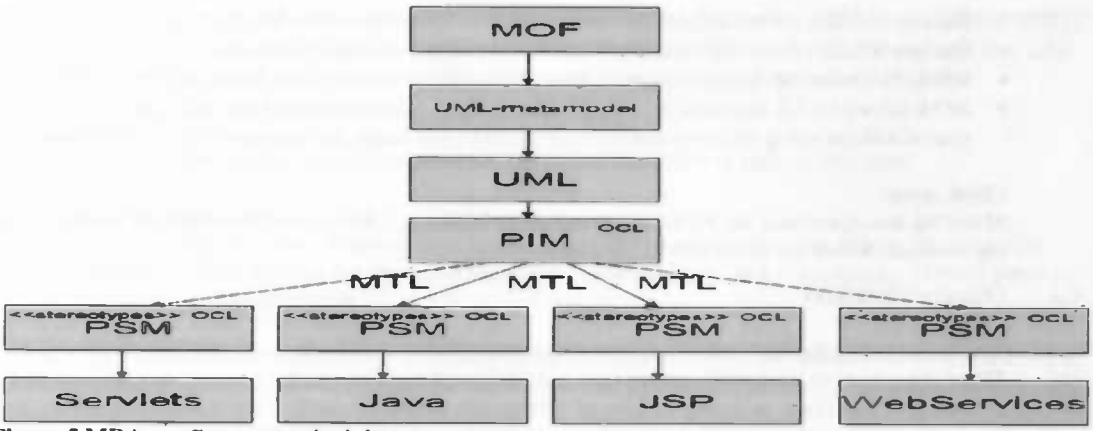

Figure 5 MDA – refinement principle

4.1.1 MDA properties and design decision

Properties:

- The Meta Object Facility (MOF) allows for picturing the world as a set of objects.
- Software development through modeling.
- UML meta-model can be used for modeling objects.
- A platform-independent model or **PIM** is a model of a software or business system that is independent of the specific technological platform used to implement it. System functionality may first be defined as a platform-independent model (PIM) [32].
- A platform-specific model **PSM** is a model of a software or business system that is linked to a specific technological platform (e.g. a specific programming language, operating system or database). Platform-specific models are indispensable for the actual implementation of a system [31]. The combination of PIM/PSM allows for a refinement of a software architecture thereby suppressing irrelevant detail.
- The different PSM's are derived from one central PIM through a MTL.
- The generation of code to the different layers of the application through there associated PSM's from one central PIM keeps the variability of the total application in sync with this PIM.
- Increased maintainability and consistency of the application through the *refinement principle*.
- The usage of <<stereotypes>> and {OCL-statements} allows for adding functionality to the PSM's and PIM.
- MDA allows for a generic way of solving the unreliable connection properties [NFR01] without putting too many constraints on the thin client approach i.e. it allows for a manageable thick client approach.
- MDA allows for code generation. Generated code is bug free, or errors are reproducible!.
- MDA defines an approach to system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform [22].
- Phenomena in the real world like eCRM can be modeled in the PIM.
- MDA allows the development of other meta-models.
- Flexibility with respect to the underlying platform specification i.e. replacing java with C# for example doesn't have a great impact.
- MDA can be used for system specification at design time.

Design decision:
We are going to use MDA to develop the eCRM object-structure through the UML modeling language.

Rationale:

- The use of MDA allows for platform independent development of the UI [NFR02].
- The use of MDA increases the maintainability of a large software application.
- MDA allows for the flexible design of the eCRM object-structure in the business logic [NFR03].
- MDA allows for UI, *interface logic and business logic* code generation from one model, this makes them related without using sophisticated decoupling tricks and keeps the application in sync with this PIM model.

Challenges:
Modeling non-functional variability requirements in UML i.e. MDA is used for modeling the functional requirements which are derived from use cases.

4.2 Object structure(s)

An object is an individual unit of run-time data storage that is used as the basic building block of programs. These objects act on each other, as opposed to a traditional view in which a program may be seen as a collection of functions, or simply as a list of instructions to the computer. Each object is capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent little machine or actor with a distinct role or responsibility [33].
An object-oriented system is composed of objects. The behavior of the system is achieved through collaboration between these objects, and the state of the system is the combined state of all the objects in it [33].

Properties:
- Can be modeled.
- Lot of standards and tools
- Representing State full things to the client which allow for easy manipulation.
- Objects hold their own context information
- Objects have information through state transitions.
- Easy extension of the business logic with new functionality
- Object-interface encapsulates its implementation
- Object-interfaces and WS-Interfaces suggest a tight relation[4] between them.
- Objects can be used for implementing both the UI and business logic.

Design decision:
To fulfill the [NFR03] i.e. functional flexibility we are going to use an object-structure that represents the eCRM functionality in the business logic and UI objects in presentation layer for manipulating objects.

Rationale:
- Object with their state, identity and behavior properties allow for easy expansion of eCRM functionality in the business logic [NFR03].
- Objects can be modeled using MDA.
- Object manipulation is intuitively clear to users.
- Keeping track of the object-state could help solving the unreliable connection [NFR01].

Challenges:
How to decouple the UI and business logic?

4.3 Twin Object principle

[4] Easy to derive a WS-Interface (wsdl file) from a java-class.

The Twin object is a principle/idea to be used to further increase the maintainability of the application [NFR03] and offer customers an intuitive way of using the eCRM functionality [NFR10]. We have the vision that users are intuitively good at manipulating objects [30]. The Twin Object principle is explained below.

Definition twin:
A twin is an entity that consists out of two parts i.e. the UI and the business logic in this case.

Definition twin object:
A twin object is an object in the UI and an object in the business logic that is generated from the same PIM model; this property directly links the objects in the UI and business logic i.e. a 1:1 mapping (mirror) between UI objects and business logic objects.

The MDA approach allows for refining the application (refinement principle) by deriving the different PSM's for the different layers through a MTL from the same PIM. The Twin object idea is a further refinement of this MDA approach.
The model of the Twin object is identical at the PIM level, at the PSM level however these models will differ through the addition of <<stereotypes>> and {OCL-statements}. To keep the differences between the PSM level models as small as possible (they can only be transformed from the same PIM if they are similar), it is preferable that most of the commonalities between objects in the business logic and its twin-objects in the UI are modeled in the PIM.

Twin object properties:
1. The twin object is a *multiple refinement principle* i.e. a further refinement of the *refinement principle* (see section 4.1).
2. The twin objects in the UI (PSM) and business logic (PSM) are generated form the same PIM model, therefore they are strongly linked.
3. The generation from the same PIM model links the objects in the UI and business logic but they are not identical.
4. The PSM models for the UI and Business logic will differ through the addition of <<stereotypes>> and OCL statements.
5. The twin object is reflexive (mirror) between UI elements and business logic properties i.e. there is a 1:1 mapping between UI and business logic.
6. The twin object increases the consistency of an application which increases the maintainability of the application.
7. Direct manipulation of the business logic through its twin in the UI thereby increasing the consistency between these layers.

Design decision:
To increase the maintainability aspect of [NFR03] and increase the usability [NFR10] we are going to use the Twin Object principle.

Rationale:
The multiple refinement principle of the Twin Object is expected to increase the maintainability of the application [NFR03]. The Twin Object principle is expected to have a positive effect on the usability of the application [NFR10] through the direct manipulation of the business logic through its twin in the UI.

Challenges:
The modeling of the Twin Object principle is expected to be a challenge i.e. the different PSM's have to be derived from the same PIM through a specific MTL, how can this be done? In section 5.5 this is analyzed in more detail.

4.4 SOA

Service orientation describes a software architecture that defines the use of loosely coupled software services to support the requirements of business processes and software users. A SOA can also be regarded as a style of information systems architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services. These services inter-operate based on a formal definition (or contract, e.g., WSDL) that is independent of the underlying platform and programming language [34].

Properties:

- Loose coupling of client/server applications
- Flexibility
- Extensive enough to express business scenarios i.e. eCRM services (eCRM functionality is not the right word in this context)
- Adaptable to changes
- Separation of concerns
- Independent of QoS
- Attached if necessary (sort of plug-in structure for QoS)
- Recoverability
- Transactions that failed can be rolled back
- Context-independent (stateless in not the right word)
- A SOA increases Interoperability of the different systems through loose coupling.
- SOA has been standardized through the implementation of Web Services standards.
- A SOA can be implemented by Web Services.
- Independence of location and implementation.

Design decision:

- The link between the business logic and the UI is going to be done through the interface logic with SOA.

Rationale:

- SOA has been standardized through Web Services which is backed up by a lot of major ICT companies (IBM, Microsoft etc).
- Loose coupling [NFR08] is influenced positively by using SOA.
- Easy composition of new Functionality [NFR03]
- Take advantage of the Web Service popularity with its recent development of for example WS-Security, WS-ReliableMessaging etc. These frameworks/standards make QoS mechanisms possible [NFR06] for handling communication between the interface logic and the front-end logic.

4.4.1 Web services Framework

The most popular implementation of a SOA is done through Web Services. Below a framework of available standards within the Web Services world is given.

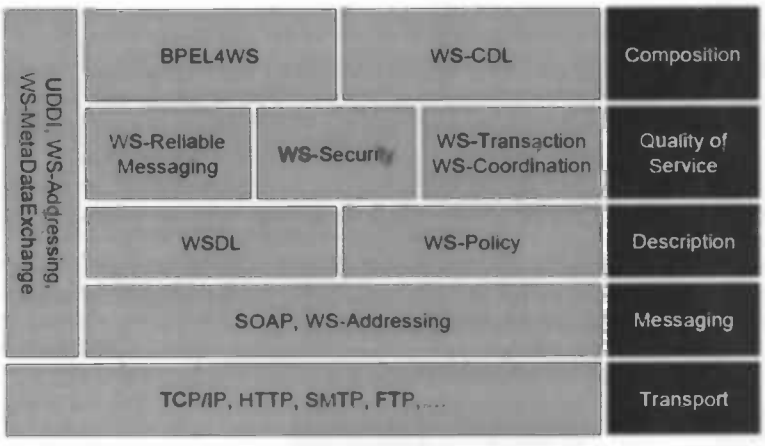

Figure 6 Web Services Framework

Properties:

- The actual interface of Web Services is done through WSDL files. WSDL interfaces are highly adaptive compared to object interfaces (see section Appendix A for a template of a WSDL file).
- The functionality of a Service is implemented by some application code independent of a specific language.
- Application code + WSDL interface = Web Service.
- The standardization through Web Services makes communication possible through SOAP messaging.
- Lots of available transport protocols for transmitting SOAP i.e. location independent.
- Transport overhead because SOAP is based on XML
- Uniform wsdl interface to the outside world
- Platform + language independent
- A Web Service can be addressed through an URL or through a more complex/sophisticated scheme like WS-Addressing.
- Web Service functionality can be easily composed into new functionality.
- Web Services are stateless, though policies can be applied to make them Stateful.
- The implementation of a SOA through Web Service standards increases the Interoperability of a distributed system.
- Standardization (in general) increases and loose coupling are important concepts to increase the Interoperability [NFR08] of a distributed system [21].

Design decision:
The interface logic is going to be implemented through Web Services.

Rationale:

- Most popular implementation of a SOA.
- Most legacy systems can be extended with a WSDL interface. This might be handy in case of a preferred short hack opposed to a complicated business logic interaction [NFR12].
- Standardization increases the Interoperability [NFR08] of a system.
- Web Services can loosely couple the UI and business logic.

Challenges:
The use of twin objects in the business logic and UI and its communication (interface logic) through Web Services suggests that their will be mismatch between state full objects and stateless web services. It is expected that Web services are not expressive enough to express state full interactions.

4.5 Thin and Thick client

Within the field of thin and thick client there are a lot of different definitions and intermediates. This section does not intend to give a precise definition of the different approaches but aims at judging the properties of a thin/thick client against the given requirements from section 2 and the maintainability of the application. The Future Work in section 8.3 can be consulted for more comments on intermediates like rich clients.
The picture below sketches the subdivision between the different layers for a thin and thick client as is done for the analysis in this section. The picture that is used shows an ultra thick client, this is an outdated approach but for an analysis it was found useful.
The presentation layer exists out the presentation logic and presentation manager, the application layer exists out of the business logic and the data layer exists out of the database logic. The arrows indicate which parts of the 3-Tier architecture are located where i.e. client or server side.

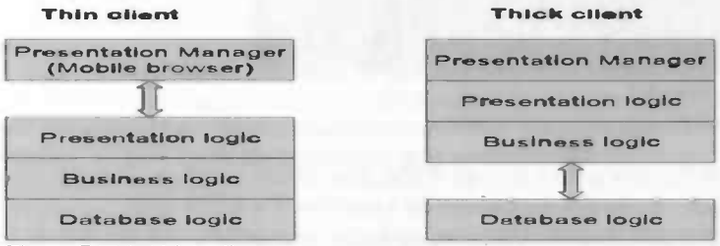

Figure 7 Thin/thick clients

Thin client:
A thin client is a computer that primarily depends on the server for doing the computational duties. The best known example of a thin client approach is the browser (presentation manager).

Thick client:
Thick clients do their own data processing and only communicate with a server for permanent storage of data.

Properties thin client:

- Thin clients increase the maintainability and flexibility of the application with respect to addition of new mobile phones [NFR02] because the applications state can be managed from one central point i.e. the server.
- A thin client approach is preferable opposed to a thick client when a requirement is that a lot of different clients need to be served. The thin client approach only requires code on the server side to serve all clients [NFR02] [36].
- Most (new) mobile phones will have some type of simple or advanced browser available that will understand markup languages like WML or XHMTL-MP [12], this also has a positive effect on the maintainability because it allows for one central application which can be used by all mobile browsers [NFR03].
- A browser allows for good intuitive navigation by a customer [NFR10].
- The use of a browser is good for Interoperability [NFR08] by using standards like WAP.
- The fact that most mobile phones will have some sort of browser is also good for the addition of new mobiles [NFR02] i.e. no specific presentation logic code for the different UI's of new mobiles.
- Another good point for the thin client is that new eCRM functionality can be added at one central place [NFR03] (able to serve many clients with only server side coding).
- A thin client cannot do any offline work; this puts a constraint on being able to handle the intermittent connectivity properties [NFR01].

- A thin client cannot do any offline work; in combination with the intermittent connectivity properties of the mobile network solutions functional solution can place a constraint on the performance of the application [NFR09].
- Some eCRM functionality might require a lot of back and forth communication by the client and the server this is bad for performance [NFR09].
- Due to the large variety in available eCRM functionality it might be that a thin client can't handle (some complex) eCRM functionality. It seams reasonable to assume that part of this eCRM functionality might require sophisticated programming solutions
- Long running transactions might require some sort of information storage on the client side; a thin client has limited abilities for this [NFR07].
- Doesn't take advantage of computational power on the mobile device which is bad for performance [NFR09].
- A thin client requires a server side state full solution for maintaining a synchronized state between client and server (see section 4.5.1).

Properties thick client:
- A thick client allows for fast data processing [NFR09].
- A thick client allows for offline data processing [NFR09].
- A lot of new mobile phones will have some version of J2ME available and a few mobile phones will have the ability to install native applications [12] which is good for the variety of available eCRM functionality i.e. able to serve complex eCRM functionality.
- Thick clients allows for the development of a more sophisticated UI and keypad than a browser which is good for usability [NFR10].
- Thick clients allow for sophisticated solutions with respect to long running transactions [NFR07].
- A thick client approach requires that new eCRM functionality has to be implemented on the client in a platform specific way this is bad for the easy addition of new eCRM functionality [NFR03] and new mobiles [NF02].
- A thick client approach has a lot of different available standards this is bad for Interoperability [NFR08].
- A thick client has a bad influence on maintaining a consistent state of the application i.e. it requires sophisticated synchronization between server and client; this has a negative effect on the maintainability of the application.
- A thin client requires a client side state full solution for maintaining a synchronized state between client and server (see section 4.5.1).

The table below outlines important properties of the two approaches.

	Maintainability	Performance	Functional flexibility	Usability
Thin client	++	--	++	+/-
Thick client	--	++	--	++

Design decision:
We are going to tentatively focus on a thin client approach.

Rationale:
A thin client has a positive effect on requirements [NFR02] and [NFR03]. The requirements have a direct influence on the maintainability and a higher priority then [NFR09] and [NFR07] for which a thick client is more suitable. There are also some attractive technologies like WURFL (see section 4.8) that can increase productivity and which can be used for a thin client.

Comment:

As can be seen both thin and thick client solutions have there own pro's and cons. A reasonable solution could be an intermediate like a manageable thick client or a rich thin client approach in the future work (see section 8.3) there will be given some hints for a more detailed analysis of thin/thick clients.

4.5.1 Client side and Server side State full solution

Due to the intermittent connectivity properties of the mobile connection which the application should be able to handle [NFR01] it could be that the application can get in an inconsistent state i.e. partially filled in forms could require long running transaction solutions. The thin client approach requires a server-side Stateful solution which will probably require a lot of back and forth communication to keep the client and server synchronized. This back and forth communication itself is again subject to the intermittent connectivity properties, this could have a devastation effect on the performance as a thin client assumes synchronous[5] (blocking) communication.

A thick client has more communication and client side state full solutions available for long running transactions and doing off-line work. This gives a thick client more options for handling the intermittent connectivity and long running transactions problems. Attractive options for solving this problem could be versions of J2ME which are available on most modern mobiles.

Properties

- A thin client (server side state full solution) requires complex synchronization algorithms between client and server for maintaining the state of the application; this could have a serious impact on the Performance.
- A thick client (client side state full solution) has more freedom for synchronizing the state of an application.

Design decision
A server side state full solution is going to be used.

Rationale
Due to the fact that a thin client approach is used (see section 4.5) this is required.

Comment
As can be seen from this short analysis this subject also needs more investigation chapter 8 can be consulted for more options and suggestion.

4.6 Highly Interactive applications

For highly interactive applications with a strong graphical user-interface component there are two patterns that can be applied; Model View Controller (MVC) and Presentation Abstraction Control (PAC). The two approaches have a very different structure in organizing the UI, Business Logic and the decoupling between these two layers through control mechanisms. Another pattern called publish/subscribe or observer pattern is closely related to MVC and will also be discussed.

4.6.1 Model View Controller (MVC)

The Model View Controller (MVC) pattern is a very popular in designing middle and large sized Web applications. In MVC, the interactive application is split up in three parts i.e. the Model contains the business logic, the View contains the displays the user-output information and the Controller handles the user input.

[5] Note: Asynchronous options are available i.e. rich client solutions like AJAX (see chapter 8.3).

The user interface is comprised out of the View and the Controller, both the Controller and the View are decoupled from the Model [24] by using a publish/subscribe mechanism. This decoupling allows for flexibility with respect to changing and adding functionality to the UI and business logic independently.

The Model component contains the business logic (processing) of the application and is linked to the View/Controller through a publish/subscribe mechanism i.e. it notifies the View/Controller about changes of its state.

The View can query the Model about changed data after receiving a notification and display these changes accordingly. Views often offer functionality that allows controllers to manipulate the display; this is useful for functionality that is independent of the Model like scrolling.

The Controller handles the input from the user and translates it into requests for procedure calls on the Model; this decouples the user input issues from the business logic. The Controller can also query the Model after receiving notifications and change its state accordingly.

Properties:

- Separation of concerns by dividing an application into three areas: processing, input and output.
- Two main layers: presentation (UI) layer (View + Controller) and business logic (Model).
- Publish/subscribe mechanism between View and Model.
- Publish/subscribe mechanism between Controller and Model.
- Decoupling of UI and business logic through the publish/subscribe mechanisms, but still a direct link between UI and Business Logic i.e. View directly queries the Model (business logic) for updates.
- MVC is asymmetric between user input and output i.e. the Controller handles the input thereby strictly decoupling user input from the business logic while the View directly queries the Model for user-output updates thereby directly linking the UI and business logic.
- One way decoupling between the presentation layer (UI) and business logic. The Controller which is located in the presentation layer decouples the user input from the business logic. The View which is also located in the presentation layer however directly queries the business logic for updates.
- One to one mapping between View and Controller.
- Controller might manipulate the (direct link) View i.e. scrolling.
- MVC is used often in tiered web applications.
- Lots of available technology frameworks like springs/struts, for fast deployment.

Publish/Subscribe mechanism:
The publish/subscribe[6] mechanism which is used by MVC allows publishers and subscribers to be decoupled from each other. Publishers post notify messages without knowledge of possible subscribers; the notification message is send to the registered subscribers which can then query for possible changed data.

Properties:

- Decoupling of publishers and subscribers but still a direct link between i.e. subscribers directly query the publisher.
- Decoupling improves the scalability by allowing mechanisms like message queuing and parallel operation capabilities.
- Asynchronous messaging pattern.
- Security issues.
- Closely related to the MVC pattern.

[6] The Publish/subscribe mechanism is also known as the Observer Pattern

Challenges:
How to model control with MVC?
How to handle device variability with MVC?

Design decision:
Due to the challenges the design decision is made in section 5.6.

4.6.2 PAC (Presentation Abstraction Control)

The PAC design pattern is another approach to developing interactive software systems, like the MVC pattern this application also makes a clear distinction between the UI the functional core (business logic) and the control of the application.
PAC organizes a software system through a tree-like hierarchy of loosely coupled agents; every agent is responsible for a specific part of the functionality of the application. Agents consist out of three components: presentation, abstraction and control. The abstraction component maintains the data and state of the agent i.e. the business logic of that particular agent, the presentation is the visible behavior of the agent and the control component connects the two thereby decoupling between UI and business logic. The control component also allows the agent to communicate with other agents. This subdivision separates the human-computer interaction aspects from its functional core and its communication with other agents [24]. The PAC design pattern distinguishes three types of agents; top-level agents, intermediate-level agents and bottom-level agents.

The top-level agent provides the high-level business logic or global data model of the system on which other agents operate. The presentation component of the top-level agent may also include parts of the interface that cannot be assigned to a particular subtask [24] and which are common to the whole application. The control component of the top-level agent allows lower level-agents to manipulate the global data, it coordinates the hierarchy of PAC agents and it maintains information about the interaction of the user with the system.

The bottom-level agents provide self contained (atomic) semantic concepts which users can act on directly. The presentation component provides the UI and the access to all the functions the users can apply to it. The abstraction component maintains the agent specific data like the top-level agent, but no other agents depend on this data [24]. The control component acts as an adapter between the presentation and abstraction. It maintains the consistency, decoupling between the abstraction and presentation. The controller communicates user events/data to the presentation or abstraction component and it communicates with higher level agents [24].

The intermediate agents can fulfill two different roles: composition and coordination. The composition allows for a new abstraction level of the application i.e. composes lower-lever agents to a single unit of higher abstraction [24]. The coordination role takes care of keeping the bottom-level agents consistent.

Properties:
- Object oriented design pattern.
- The use of a hierarchy of agents allows a system to be decomposed into different levels of abstraction.
- Separates the concerns of a system by breaking it down into loosely coupled agents.
- Both a horizontal and vertical decomposition of a system through a hierarchy of agents and the decoupling of presentation, abstraction and control components.
- Clear separation (very decoupled) between business logic and UI i.e. they only communicate through the controller.
- The tree-like hierarchy of PAC allows flexible composition. Making complex user interfaces based on simple components without violating encapsulation of the UI and the business logic i.e. it is very legal that controllers communicate and collaborate with each other [23].
- PAC is symmetric between user input and user output.
- Recursive composition of control.

- The representation of the data in the abstraction component of the top-level agent is media independent, this allows for support on different platforms without major changes to the abstraction component.
- PAC scales very well, various components can be tied together in a strict decoupled way.
- The UI and business logic Components of PAC are strictly decoupled through the Controller with respect to MVC.
- PAC can take care of the dimensions of mobility and affect the user interface without exposing this functionality to the core logic of the application.
- Flexibility with respect to composition and delegation.
- Control aspect of the agents is recursive and might get complicated.

Challenges:
How to model control with PAC?
How to handle variability with PAC?

Design decision:
Due to the challenges the design decision is made in section 5.6.

4.7 Communication

The communication between the mobile client and the WEB/WAP server can be provided in two ways i.e. a direct link (the internet) through WiFi (hotspot) or by using a WAP gateway. Both options have a lot of available protocols at their disposal, WiFi uses http related protocols like TCP/IP or SOAP and a WAP gateway uses WAP related protocols like WIP/WSP[7]/WAP-push. There are however a lot of different options for choosing protocols most modern WAP gateways for example can handle SOAP.

Properties:
- HTTP uses 3-way handshaking which gives a lot of performance overhead on mobile networks.
- WIP/WSP uses a 2-way handshaking mechanism which gives better performance on mobile networks.
- Modern mobiles that have WiFi also have HTTP capabilities.
- A WAP gateway is the most standard form of communication provided for mobile devices.

Design decision:
We are going to use HTTP or WAP/WIP communication between the mobile client and the presentation logic.
We are going to use SOAP for communication between the presentation logic server and business logic.

Rationale:
A thin client requires a WAP-gateway for communication with mobiles. The associated WAP protocols are implemented on basically every mobile phone which is positive for device flexibility [NFR02] i.e. no device specific code is required for serving many different clients.
The communication between the presentation logic and business logic is going to be done through SOAP which is related to the choice of Web Services for the Interface Logic component (see section 4.4.1) the usage of SOAP (Web Services) decouples the business logic from the presentation logic which is positive for the maintainability of the application (can be altered independently).

[7] WSP is best thought of on first approach as a compressed version of HTTP which can be used for mobile networks.

4.8 WURFL/WALL

Wurfl is a run-time solution for handling the device flexibility requirement [NFR02]. The (Wireless Universal Resource File) WURFL is an XML configuration file (in Appendix A a code snippet from wurfl.xml itself is added) which contains information about capabilities and features of most wireless devices. The main scope of the file is to collect as much information as we can about all the existing wireless devices that access WAP pages so that developers will be able to build better applications and better services for the users [14]. The information that is collected through the WURFL can be used in two ways, firstly it gives the ability to use one generic presentation for the UI and secondly it can be used to run-time discover device specific capabilities like J2ME or screen resolution.

At present there are some 400 devices, each with up to 100 characteristics. In order to avoid an unmanageably large XML file, the WURFL describes a device hierarchy. This is a compact and effective description, because devices fall naturally into groups depending on their manufacturer and browser software. This makes it possible for the WURFL to make conjectures about devices not explicitly included; for example, NOKIA devices all share the same basic browser software [28].

WALL is the most used implementation of the WURFL through JAVA. WALL can handle the collection of available mobile markup languages like XHTML-MP[8] flavors, cHTML and WML-1.X as one markup language; WURFL detects runtime which markup languages are preferred by the different mobile devices and delivers the specific suitable markup presentation from the same general presentation. The picture below [14] sketches the domain of the different markup languages.

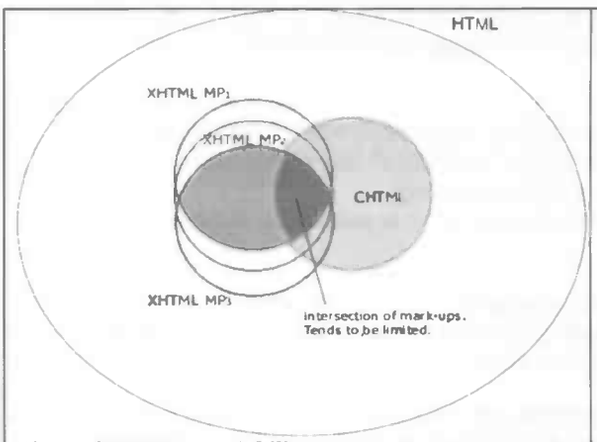

Figure 8 Markup variability

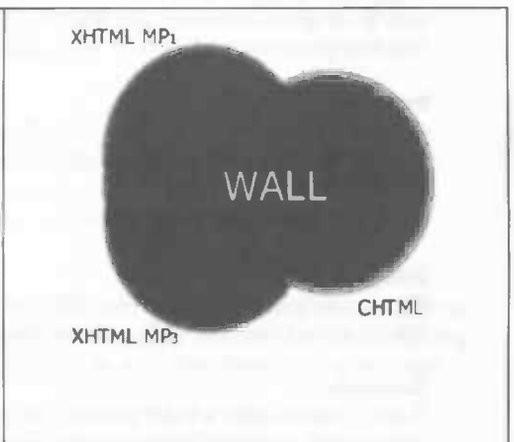

Figure 9 WALL solution

As mentioned above WURFL is basically a run-time solution for the variability in the domain of mobile devices. It abstracts away from the minor differences between the different markups by pre-defined programming solutions. Markup languages like XHTML-MP differ from HTML and cHTML in that it needs to be well-formed according to the XML syntax. For example a simple
 tag needs to be closed with a </br> tag in some flavors of XHTML-MP. WALL abstracts away from this by detecting what a device can do and make sure that the best possible mark-up is delivered by simply using the <wall:br/> tag.

The differences within the different mobile markup languages [15]:

[8] XHTML-MP supersedes XHTML Basic which is a subset of HTML.

- Most CHTML devices honor the
 tag and simply ignore
 tags
- Most XHTML-MP devices will honor both
 and
 tags.
- Some XHTML-MP devices will throw an error if a single unslashed
 is found and no information will be displayed (except a pretty useless error message)

The <wall:br/> tag will [15]:

- Look at the user-agent for you,
- Query the WURFL to figure out which mark-up the requesting device prefers, and
- Issue
 or
, depending on the capability of that device.

The example above illustrates how WURFL basically functions, in Appendix A a code snippet is added which illustrates the actual java implementation of the </wall:br> tag

The WURFL can also be used to detect device capabilities like supported picture formats, wap-push[9] or j2me capabilities in a very simple manner. The example below illustrates this in an example.

```
<c:if test="${capabilities.wap_push_support}">
  <wall:a href="http://url2" title="Sport">Real-Time Sport Updates</wall:a>
</c:if>
```

The link to the wap-push is only shown for devices which are capable of handling wap-push, for other devices the link is rendered out at run-time.

Properties:

- Suitable for jsp. WALL tag-lib is build on top of the Java-Api.
- One general presentation (View) can be used by detecting the device specific markup capabilities.
- Allows for device specific code generation [NFR02].
- Uses one xml files with "all" mobile device capabilities, i.e. not only markup differences but also properties like screen-size, caching and j2me capabilities can be queried [NFR02].
- Server side implementation.
- Thin client approach.
- Simple and fast solution.
- Device hierarchy allows for elegant handling of unsupported features i.e. if a feature is not supported on the specific mobile Wurfl falls back on the generic device capability of that specific feature.
- No guarantees are given as it is open-source and dependent of the mobile industry, this is also true for other standards/approaches.
- Open source allows for continuous updates w.r.t to different devices and bug-fixes.
- WALL allows for different views bases on one generic jsp file.
- The WURFL has become widely used in the community as the most effective and up-to-date solution to device description.
- WURFL does not have an effective API, and it is based on a non-standard one-off schema.
- Large industry support.
- Runtime mapping solution i.e. detection of device capabilities is done runtime.

XML → XSLT transcoding:

Another option for multi-serving different devices from one general presentation is the use of XML/XSLT transformations; this approach allows for multi serving by applying an xslt transformation based on the device preferred markup. For example if it's xhmtl, call the xhtml.xslt, chtml.xslt or wml.xslt to transform to the required markup.

For every page you have to write a suitable transformation and do some conditional checks to check for wml-1.1/2/3/2.0 support.

[9] WAP Push messages are sms messages that alert the user (used in Blackberry devices).

Properties:
- Generic xml template for UI elements.
- More clear separation of concerns than Wurfl/Wall.
- XSLT files become very large and are hard to understand.

Design decisions:
We are going to use WURFL to support the variability in the capabilities of different mobile devices.

Rationale:
WURFL is a much simpler solution than the XML → XSLT transformation. XML → XSLT however does provide for better separation of concerns and this property is attractive with respect to code generation.

4.9 Variation Points

Variation points are UML constructs that can be used design time (PIM/PSM) to express the differences between mobile devices. From these variation points additional device specific code can be generated to handle device flexibility like differences in screen-size, operation system etc.

Properties:
- Design time solution for device flexibility [NFR02].
- Can be applied on PIM and PSM level.

Design decision:
Variation Points are not used for handling device flexibility.

Rationale:
Variation points are design-time solutions which are used during the code generation. The different mobile capabilities however have to be detected at runtime; this makes variation points unsuitable for runtime deployment.

4.10 WML + JSP + Java

The actual implementation of the underlying execution platforms i.e. the run-time technologies that will have to fulfill the application are WML/JSP and JAVA related technologies. The Java Server Pages (JSP) code is going to be generated from the presentation PSM and the Java code is going to be generated from the business logic PSM.

Wireless Markup Language:
WML based on XML, is a content format for devices that implement the WAP Wireless Application Protocol specification, such as mobile phones. This preceded the use of other markup languages now used with WAP-2.0, such as XHTML, XHMTL-MP and even standard HTML [38]. Another markup called cHTML was developed by iMODE and is an alternative to WML. All the described markups can be dynamically generated by Jsp's.

Java Server Pages (JSP):
JSP is a Java technology which can be used to implement the web platform. Jsp's allow software developers to dynamically generate HTML, XML, XHMTL-MP or other types of documents in response to a Web client request. The technology allows Java code (through scriptlets) and certain pre-defined actions to be embedded into static content. Jsp's can be extended with tagged libraries.

Tagged library:

Tagged libraries are part of the jsp technology and allow developers to easily handle custom action i.e. they can encapsulate recurring tasks. The JavaServer Pages Standard Tag Library (JSTL) is a component of the Java EE Web application development platform. It extends the JSP specification by adding a tag library of JSP tags for common tasks, such as XML data processing, conditional execution, loops and internationalization [39]. The JSTL is the most used tagged library and can be used in combination with WURFL/WALL.

Scriptlets:
Scriptlets can be used in Jsp's thereby allowing core Java code to be incorporated in dynamic web pages.

Java:
Object oriented programming language.

JavaBeans:
Object classes programmed in Java which are a lot in web applications. Java Beans are persistent during a session. A web session encapsulates a Stateful interaction of a specific client with a web server for a predefined amount of time.

Design decisions:
- Java is going to be used for implementing the business logic at design time.
- Jsp's and JSTL are going to be used to dynamically generate the UI (markup languages) and the presentation logic.
- JavaBeans and scriptlets are going to be used as part of the interface logic to handle/forward client requests.

Rationale:
- The above technologies are closely related to each other and are a proven method for implementing dynamic WEB/WAP pages.
- Jsp's can be combined with WURFL by using the JSTL.
- JavaBeans and scriptlets can be easily used in combination with jsp's.

4.11 Servers (Glassfish)

For the communication between the UI and the business logic we need to transform the http requests from the mobile browser to soap requests for the business logic. Two possible solutions for this are Axis2 and Glassfish. They are both based on Java. In this section only the glassfish server will be described. Glassfish is the name of an open source development project for the next generation Sun Microsystems Java EE server.

Properties:
- Standalone web server.
- Standalone application server.
- Able to deploy web services.
- Able to deploy servlets and jsp's.
- Uses *wsimport* (build in tool) to create client side stub for the interface logic.
- Claims to generate 85% less code than axis for handling web service interfaces.
- Uses a simple @ construct to signal the use of a web service.
- Includes an implementation of WS-addressing.
- Incorporated JAX-WS library which allows for simple Web Service deployment and communication.

Design decision:
We are going to use glassfish for the deployment of the presentation layer, interface logic and business logic.

Rationale:
It generates much less code than axis2 which is preferable for a Proof of Concept.

4.12 1 to 1 mapping Java classes – WSDL interface

Every java class can be turned into a Web Service through the addition of a generated WSDL file.
The use of glassfish makes it possible to easily create a 1 to 1 mapping between WSDL and a suitable java class.

Note: A nice property of Web Services is that they can easily be composed into new Web Services thanks to the descriptive nature of the WSDL interfaces. This property of Web Services is not used if we create a Web Service for every java class.

Properties:
- 1 wsdl interface for every generated java class.
- 1 wsdl + 1 java class = 1 web service.
- The composition property of web services is not used.

Design decision:
We are going to use a 1 to 1 mapping between java classes and WSDL-interfaces.

Rationale:
For the proof of concept using a 1 to 1 mapping in combination with the usage of wsimport (see previous section) simplifies the creation of the PSM and PIM models. I.e. the interface logic does not require a specific PSM but can be indirectly generated from the business logic PSM.

4.13 UML modeling and Code generation tools

This section does not give a thorough analysis but just gives some hints for choosing UML modeling and Code Generation tools, more thorough investigation on this subject is left to the reader.

UML modeling tools:
For modeling the PIM/PSM in UML tools like Visio or Poseidon can be used.

Code generation tools:
The ultimate step in the MDA approach is the generation of code which can be executed at runtime by the application. Some code generation tools are AndroMDA, ArcStyler, OptimalJ and EMF.

Design decision:
We are going to use Microsoft Visio for the UML modeling. The actual code generation is going to be done by human interpretation.

Rationale:
- From a practical point of view i.e. experience with Microsoft Visio.
- For the Proof of Concept human interpretation leads to faster results than the actual implementation of code generators which are required for using the suggested MDA approach. For the human interpretation we only have to show that nothing contradicts and that it indeed could have been generated.

Comment:
AndroMDA appears to be the most complete and most used tool for the MDA approach. AndroMDA includes Poseidon for UML modeling.

5 Challenges

In this chapter an analysis of the challenges that were formulated in the previous chapter is done. If applicable the analysis of a challenge is finalized with a design decision. The design decisions from the previous chapter and from this chapter together constitute the development of the elaborated model (see section 5.8) which will be tested in the Proof of Concept from chapter 6. The list of challenges that are analyzed is described below.

Section	Title	Proposed solution
4.1.1	Modeling non-functional requirements with MDA	None
4.4.1	Stateful Object/Stateless Web Service mismatch	Generation of control methods in the business logic which express state full information. Use of (state full) web service standards like WS-RF or WS-Context
4.1.1	How do we model the variability in the different layers	Within the MDA approach the PIM can be used to Model eCRM variability (functional flexibility) at design time and WURFL can be used to handle device variability (variability in capabilities) at run-time.
4.2	Decoupling of the presentation logic and business logic	Combination of interface logic and PAC or MVC
4.3	Modeling the Twin Object in the UI and business logic.	Addition of <<stereotypes>> and {OCL-statements} to the UI and business logic PSM.
4.6.1 4.6.2	How to model Control using PAC/MVC	None
4.6.1 4.6.2	How to handle variability using PAC/MVC	None

5.1 Modeling non-functional requirements with MDA

MDA is used for modeling the functional requirements and not for non-functional requirements like screen-size and operating system of an application. This challenge however is no longer valid due to the fact that WURFL has been chosen for support of device flexibility [NFR02].

5.2 Stateful Object/Stateless Web Service mismatch

The use of twin object in the business logic, UI and its communication through the Interface Logic component (implemented by Web Services) suggests that there will be a mismatch between state full objects and stateless Web Services.

Hypothesis: Web Services are not expressive enough to model Stateful interactions

Proposed solutions:

- Use WS-RF to create a Stateful resource.
- Use MDA to model getter and setter methods for accessing state full information.

Comment:

The proposed solution for this challenge has been moved to Future Work (see chapter 8).

5.3 How to model the variability in the different layers

This challenge was developed in the beginning of the internship. The proposed solution is basically the development of the elaborated model. The eCRM functionality can be modeled in the PIM, WURFL can be used in the presentation PSM to handle device flexibility and the addition of stereotypes and ocl-statements is assumed to be sufficient for the implementation of the Twin Object Principle thereby being able to create the different PSM's from only one PIM.

5.4 Decoupling of the UI and business logic

This challenge was also developed in the beginning of the internship and came from the idea that we wanted to present Stateful things to user which allow for easy manipulation. The suggested decoupling is required in the MDA approach for being able to generate the code for both the business logic and the UI PSM's independently. The proposed solution for this is done through the usage of the Interface Logic and a choice (see section 5.6) between the PAC/MVC patterns.

5.5 Modeling the Twin Object principle in the UI and business logic PSM's.

For being able to generate from one PIM to the different layers of the application it is required that the PSM's are quite similar. To keep the two PSM's similar the Twin Object principle was proposed. The Twin Object principle envisions "ideally" a 1:1 mapping (mirror) between objects in the business logic and it presentation object on the screen. It is expected that the addition of <<stereotypes>> and {OCL-statements} are strong enough to overcome the differences between these two PSM's and thereby allowing that both PSM's are generated from the same PIM. The specific required <<stereotypes>> and {OCL-statements} will be developed in the Proof of Concept (see chapter 6).

Proposed solution:
The refinement of the MDA approach with the Twin Object principle and the addition of <<stereotypes>> and {OCL-statements} is strong enough for being able to generate the code in the different layers and still allow the associated PSM's to be derived from the same PIM.

Note: The Twin Object principle property of being able to derive the application from one PIM is required for being able to keep the variability in sync with this PIM.

5.6 How to model control in the application

As MDA is going to be used to increase the applications maintainability, it is important that all aspects of the application can[10] be generated. If all aspects of the application have to be able to be generated it is required that all aspects of the application can be formalized in a "simple" model(s) through general constructs.
For a model to stay simple it is required that it is decomposed into manageable pieces. Design patterns are a way of decomposing an application into manageable pieces. In section 4.6 we discussed two design patterns MVC and PAC. Both patterns can be used for the decomposition of a highly interactive large application with a strong user-interface element.
Another factor that plays an important role in the suitability of one of these patterns is that there will be a lot of variability with respect to the differences in Mobile devices [NFR02].

[10] It is not required that all aspects have to be generated i.e. in some cases it is allowed to add manual code.

Both MVC and PAC provide a separation of concerns (decomposition) between UI and business logic. The properties of these patterns however differ i.e. MVC focuses on a separation of concerns by dividing an application into processing/input/output and PAC divides an application into presentation/abstraction/control. In this analysis some of the properties and their effect on being able to be formalized in model are investigated. This analysis focuses on the control issues i.e. the glue between the presentation logic (UI) and the business logic, the design decision is formulated in section 5.7.1.

Control has been defined as "the link between syntax and semantics" [26]. The role of the control part is to maintain an extensive knowledge about the two worlds it serves.

5.6.1 Control analysis MVC/PAC using one View.

MVC:
MVC decomposes an application in three areas, processing, input and output [24]. The result of this subdivision is that in MVC there is no clear notion of what control is [23] the notion of control is diluted across the Model/View/Controller objects and is not handled at one specific place.
The decoupling between the UI and business logic is done through the publish/subscribe mechanism, although this indeed decouples between the two there is still a direct link (one-way decoupling) between the View and the Model i.e. the View directly queries the model for update information. It is unlikely that changes to the Model or View can be generated independently [24].
MVC is asymmetric between input/output this allows for flexibility in treating input and output control issues separately but it is also the case that a lot of input events are directly coupled to output events, for example scrolling.

Due to the unclear notion of control, the one-way decoupling and the asymmetry between the input and output MVC causes that the notion of control is handled in all three components. The one-way decoupling tightly couples the UI and business logic. This leads to a monolithic control layer with complex dependencies and will require a lot of handwritten code which cannot be generated from a simple model; this is illustrated in Figure 10.
The proliferation of Views which could be a consequence of handling the device variability adds to this problem (see 5.7.1).

PAC:
In PAC the notion of control is directly located in the controller component. The usage of a separate control components allows for a very strict decoupling of the UI from the business logic. The strict decoupling between the UI and business logic make it suitable for the composition of complex interfaces without violating encapsulation and making both the presentation layer and business logic suitable for formalization in a model. Within the presentation component input and output is treated symmetrically, as many input related events are closely related to direct output this is not a real disadvantage.

The advantage of PAC with respect to model driven development is exactly this central place for handling the control thereby strictly decoupling between UI and business logic and allowing for handling the different aspects independently. As PAC is basically a recursive pattern, controllers can be placed into other controllers and control can be handled as small chunks which can be modeled. This modeling can be done through the development of dialogues.

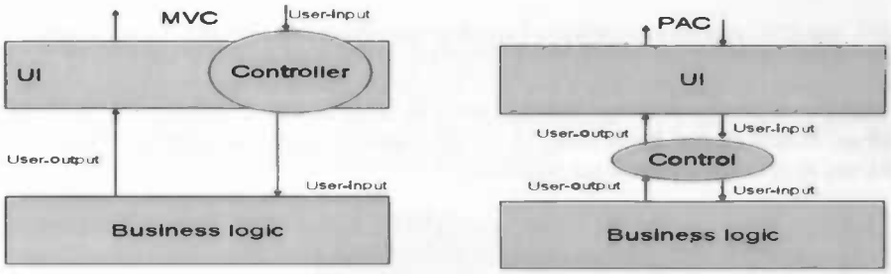

Figure 10 MVC versus PAC

5.7 How to handle device variability?

Device flexibility must be handled runtime by using one or multiple views representing the eCRM functionality. In section 4.8 we decided to use WURFL which is based on one generic View for handling device flexibility [NFR02]. Another option that can be considered is creating different views between which can be switched for different devices. The section below gives a short analysis of how PAC and MVC handle control for multiple views.

5.7.1 Control analysis PAC/MVC using multiple views.

MVC:
In MVC there is a 1:1 mapping between View and Controller. The variability in the number of mobiles [NFR02] and networks [NFR04] could mean that it will be necessary to create a lot of different Views with associated Controllers. This expansion in Views and Controllers becomes unmanageable through the close coupling between View and Controller and the direct link with the Model [23]. Another negative side effect is maintaining the consistency between all these different views and controllers, for example new functionality has to be added at exactly the same point in the different Views and Controllers [23]. Both these problems add to the notion of the already existing problems as mentioned above i.e. the creation of a monolithic control layer.

PAC:
Variability in mobile networks and devices could require the development of new functionality just to handle the switching between different views. The flexibility of PAC with respect to composition and delegation of this specific switching functionality it's possible to scale up PAC so that some PAC agent handles just this type of control without interfering with the PAC pattern itself.
By applying PAC recursively at every level of abstraction of the user interface, everything in an interactive application is a PAC object [25] which can be formalized in models.

Other Concerns:
- The PAC pattern allows for handling QoS or low-battery level problems i.e. dimensions that come with mobility to be handled by separate agents [23], although these mobility aspects are not part of the direct requirements this is also an advantage.
- PAC hierarchy allows for different levels of abstraction itself i.e. the business logic could be seen as the PIM.
- MVC requires the MDA approach to create this refinement in abstraction.

Design Decision:
PAC is going to be used for handling the decoupling between presentation layer and business logic.

5.8 Elaborated Model

The list below shows a summary of the made design decisions from this and the previous chapter. The summary is graphically illustrated in Figure 11.

Design decisions summary:

1. 3-Tier architecture
2. Interface logic implemented by Web Services.
3. MDA is used for modeling the design time variability.
4. Front-end server for implementation of the presentation logic through Jsp's.
5. Back-end server for implementation of business logic through Java(Beans).
6. UI elements represented through WURFL/WALL→ XHTML/WML/cHTML to handle device flexibility [NFR02] (a one view approach).
7. XHTML-MP/WML/cHTML indirectly provided through jsp's and WALL
8. Both UI elements and control elements are modeled through the Presentation layer PSM.
9. Control elements are implemented by scriptlets.
10. WSDL interface for the interface logic indirectly provided by glassfish i.e. indirect code generation for the WSDL-interface.
11. 1:1 mapping of wsdl interface and java object of the business logic.
12. A thin client approach WURFL/WALL for handling device variability.
13. PAC design pattern for handling decoupling between presentation layer and business logic.

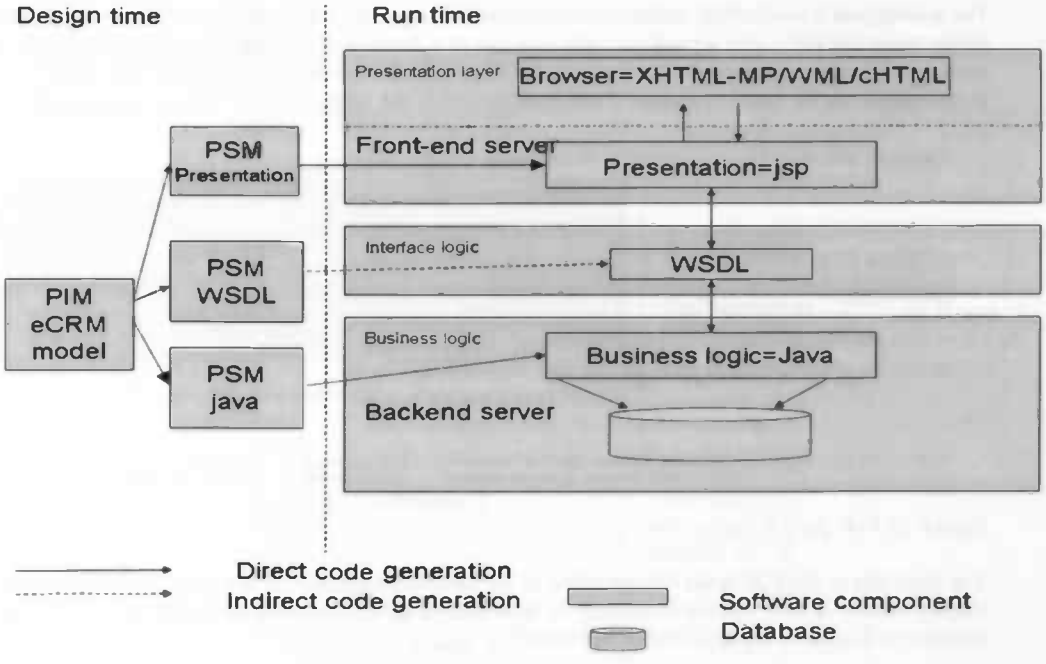

Figure 11 Elaborated Model

6 Proof of Concept

The Proof of Concept (PoC) is used to reflect on the elaborated model from section 5.8. The reflection will consist out of an analysis of the implementation of the elaborated model using the Twin Object principle. To simplify the PoC the elaborated model has been split up in three parts; the first part constitutes the elaborated model without the interface logic, its related PSM and the usage of WALL. Part 1 will function as the focus for this chapter and is illustrated in the figure below.

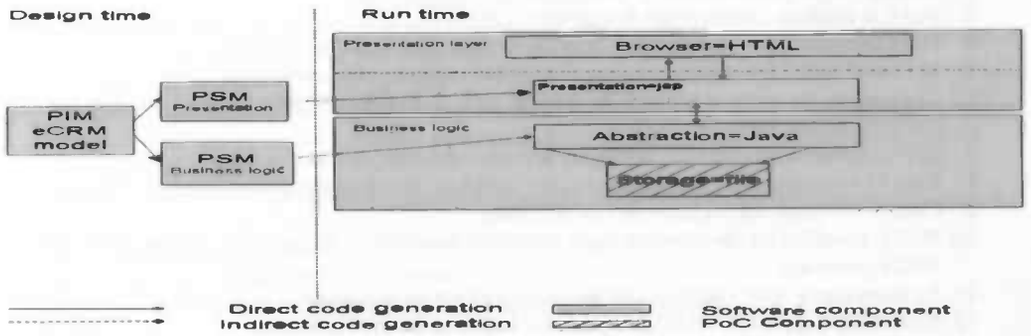

Figure 12 PoC part 1

The second part from the PoC will try to incorporate the usage of WALL to support the device flexibility requirement [NFR02]. The second part will exist out of a theoretical investigation of the (expected) required source code and its effect on the presentation PSM; no actual implementation will be done. The theoretical investigation can be found in chapter 7 and is illustrated in the figure below.

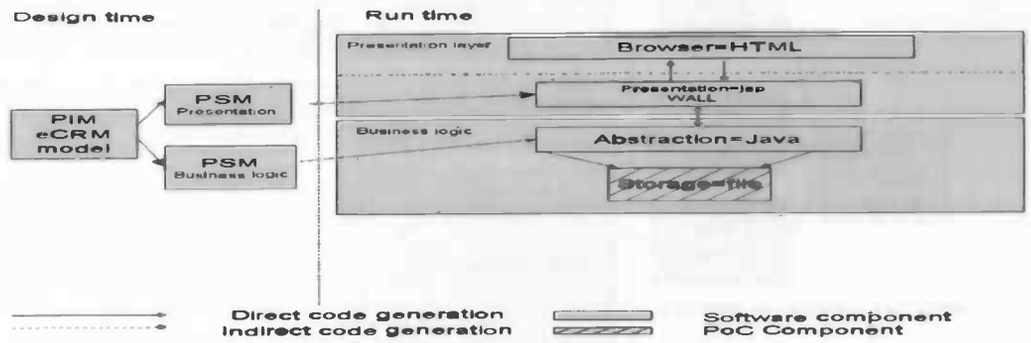

Figure 13 PoC part 2

The third part of the PoC is the incorporation of the Interface logic and its related PSM. In the Future Work chapter 8 hints on how this can (expected) be done will be given. The third part completes the PoC and is identical to Figure 11 from the elaborated model.

Some other differences between the elaborated model and the Part 1 of the PoC are:

- The business logic is placed on the front-end server instead of the back-end server; this is related to the removal of the interface logic.

- The Jsp's that implement the presentation produce HTML instead of XHTML/WML/cHTML; this is related to the removal of WALL. The HTML elements used in the PoC are so basic that they have an equivalent in the wireless markup languages.
- The storage is done through Serializable files instead of a separate database. This can be done if the permanent data storage is very basic.

The PoC will be developed and analyzed through the engineering of a few use-cases in the context of a shopping cart like example. A shopping cart (can) is part of the eCRM functionality i.e. it is related to the Selling part of eCRM as illustrated in Figure 1.

The work approach that is going to be used in the PoC will consist out of a (mainly) top-down development. Firstly some shopping cart related use-cases will be developed. Secondly these use-cases will be analyzed, application objects will be identified and an UML-model (base PIM) of the application objects will be developed. Thirdly a graphical representation of the UI will be developed. Fourthly an analysis of the required stereotypes and ocl-statements for fulfilling the functionality at the PSM level will be done. Fifthly the PIM will be improved by applying the identified ocl-statements from the PSM analysis. Finally a source code analysis will be done.

Schema work approach:
1. Use case development + analysis of the leading shopping cart example.
2. Development of an UML model the PIM.
3. Development of a graphical presentation of the UI
4. Analysis of the <<stereotypes>> and {ocl-statements} required by the business logic PSM + presentation layer PSM.
5. Improve the PIM
6. Source code analysis

6.1 Use Cases overview

In this section a few use-cases in the context of a shopping cart are developed and analyzed. The development/analysis of the use cases will lead to the development of an (UML) PIM model (see Figure 16). The PIM model will function as the basis from which the related PSM's (see section 6.8) are analyzed and developed; Figure 14 gives an overview of the use-cases that will be used to construct the shopping cart functionality. The use-cases are graphically illustrated in section 6.7.

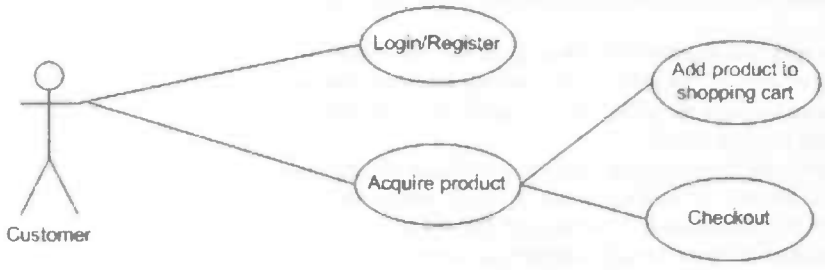

Figure 14 use case overview

6.2 Use case 1: Login/Register

The login/register use case is graphically illustrated in Figure 17 (page 1, page 2, and page 3).

Actors: Customer (C), System (S)

Description Login authorization:

1. C: a user navigates to a main page with a login button
2. S: the browser serves up a page which includes a login button
3. C: the user requests a login by pushing the login button
4. S: the browser asks for a user name and a password
5. C: the user fills in his user and password details and submits his details
6. S: the system evaluates the user details
7. S: upon a successful evaluation the system serves up a page which replaces the login button with a logout button, and it displays a textual error message in the present page upon an un-successful evaluation.
8. S: upon a successful evaluation the user is now logged and known to the system.

Description Logout authorization:

1. C: a logged user pushes the logout button
2. S: the system logs out the customer and the login button will appear.

Description Register authorization:

1. C: navigates to the register page by clicking on a register button on a page.
2. C: clicks on the register button.
3. S: interprets the browser request and constructs the register-page.
4. C: fills in the register form and pushes the send button.
1. S: interprets the request and processes the register-request-data for permanent storage.
2. S: The customer is now known to the System and a suitable response page with a login possibility is constructed.

Analysis login/logout:

C: a user navigates to the main page with a login/logout buttons.

S: the browser serves up a page which includes a login or logout button

- Users can be either logged in or logged out.
- Login, logout requests are represented by a button.

C: the user requests a login by pushing the login button

- The login request opens a separate page.

S: the browser asks for a user name and a password

- A login request requires the ability to represent parameters and present an input field for these parameters.

C: the user fills in his user and password details and submits his details.

- A login request requires the ability to represent parameters through a textfield.
- A login request requires the ability to represent input parameter fields.

S: the system evaluates the user details

- There is knowledge about a list of customers with associated persistent customer data.

S: upon a successful evaluation the system serves up a page which replaces the login button with a logout button, and it displays an error message in the present page upon an un-successful evaluation

- The invocation of the login request changes the state of a user (logged on/logged out).
- The result of an unsuccessful invocation of the login request is presented through a textfield.

C: a logged user pushes the logout button

- The logout request is presented through a button.

S: the system logs out the customer.

- The invocation of the logout request changes the state of a user (logged on/logged out).

Analyses register:
C: navigates to the main page with a login and register button.
C: the user requests a register by pushing the register button.
- Unlogged users can register.
- Register request is presented through a button.

S: the browser asks for a user credentials like name, 2 identical passwords, address.
- A register request requires the ability to represent parameters and present an inputfield for these parameters.

C: fills in his credentials and pushes the send button.
- A register request requires the ability to represent parameters and input fields.

S: interprets the request and processes the register-request-data for permanent storage.
- Register parameters can be checked.
- Customer data requires being stored permanently.

S: The customer is now known to the System and a suitable response page with a login possibility is constructed.
- After a successful register a user is now logged and known to the system.

Analysis application objects:
The login use case suggests two separate application objects:
- A SecurityObject which provides the functionality for logging of users (login/logout).
- A SecurityObject that has information about a list of known and allows registering a new customer.
- A CustomerObject keeps track of known data about the customer.

The UML diagram below gives a description of the identified application objects with their methods, attributes and associations.

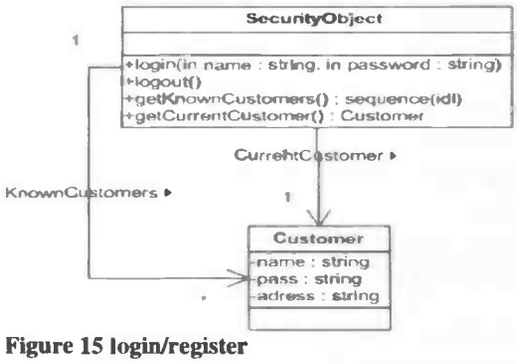

Figure 15 login/register

6.3 Use case 2: Acquire product

The acquire use can be divided into two separate use cases. For acquiring a product it is necessary to make a deal[11] between a product and a customer and it is necessary that a bill is associated with this particular deal. The starting point for the Acquire product use-case is graphically illustrated in Figure 18 (page 1).

Actors: Customer (C), System (S)

Description:

1. C: navigates to the main page which includes a list of product descriptions.
2. S: constructs a page which includes a list of product descriptions that can be navigated i.e. a link to the associated product is provided.

Note: The execution of the link is the start of the Add to shopping cart use-case.

Analysis acquire product:
C: a user navigates to the main page with a list of product descriptions.

- There is knowledge about a list of products and these individual products can be previewed.

S: constructs a page which includes a list of product descriptions that can be clicked i.e. a link to the associated product is provided for every product.

- Products can be previewed and include unique identification information and navigational information that can be executed.

Analysis application objects:
The acquire product use case suggests two separate application objects:

- A CatalogueObject that keeps track of all the products that can be sold.
- A ProductObject that keeps information about a product that can be sold on the site and can be queried for a description.

The UML diagram from Figure 16 gives a description of the identified Objects with their methods, attributes and associations.

[11] This sort of relationship represents an ownership-relation in eCRM.

6.4 Use case 3: Add product to ShoppingCart

The "add product to ShoppingCart" use case is graphically illustrated in Figure 18 (page 2). It is invoked by clicking on product links from the main page.

Actors: Customer (C), System (S)

Description: (logged customer)
1. C: Executes the unique link of a product from the list of products.
2. S: Serves up a page with "all" the product information from the unique product. If the user is logged, the page includes a button which can be used for adding a product to the shopping cart is presented.
3. C: presses the "add to shopping cart" button.
4. S: if the customer is *logged* a page in which details about the order-request can be filled in.
5. C: fills in the order request details like color and quantity and pushes the submit button.
6. S: Evaluates the correctness of the filled in order details, stores these details with the unique product and returns to the main page.

Analysis Add product to shopping cart:
C: Executes the unique link of a product from the list of products.
- A product can be uniquely identified.

S: Serves up a page with "all" the product information from the unique product. If the user is logged the page includes a button which can be used for adding a product to the shopping cart is presented.
- Product description data which is associated with a product.
- A "add to shopping cart" button depends on the logged status of the customer.
- A product is associated with a cart.

C: presses the "add to shopping cart" button.
S: if the customer is *logged* a page in which details about the order-request can be filled in.
- A product has order request data associated with it.

C: fills in the order request details like color and quantity and pushes the submit button.
S: Evaluates the correctness of the filled in order details, stores these details with the unique product and returns to the main page.
- Order request data can be checked for correctness i.e. available color and no negative quantity.

Analysis application object:
The Add to shopping cart use case suggests two separate application objects:
- A CartObject which has knowledge about which products have been already added to the cart.
- An OrderRequestObject that represents the fact that a certain product has been added to the cart and its associated OrderRequest data.
- A ProductObject that keeps information about a product that can be sold on the site. And allows adding a product to a deal.

The UML diagram from Figure 16 gives a description of the identified Objects with their methods, attributes and associations.

6.5 Use case 4: Checkout

Actors: Customer (C), System (S)

Description:
1. C: A logged customer navigates to the main page and presses the "View cart" button.

2. S: Checks if the shopping cart is not empty and shows a list of made deals (contents of the shopping cart, with ordered products). It includes a "Finalize" button in its page.
3. C: Customer pushes the Finalize button.
4. S: constructs a billing page, which includes the details of the logged customer and parameter fields that can be used for filling in additional data required by a bill.
5. C: pushes the submit button.
6. S: Finalizes the bill i.e. it is paid and return to the main page.

Analysis checkout:

C: A logged customer navigates to the main page and presses the "View cart" button.

- Only a logged customer can View a shopping cart.

S: Checks if the shopping cart is not empty and shows a list of made deals (contents of the shopping cart, with ordered products). It includes a "Finalize" button in its page.

- A cart can be checked if it is empty.
- A cart can be queried for displaying data about deals.

C: Customer pushes the Finalize button.

S: constructs a billing page, which includes the details of the logged customer and parameter fields that can be used for filling in additional data required by a bill.

- A cart can have an associated bill with it.

C: pushes the submit button.

S: Finalizes the bill i.e. it is paid and return to the main page.

- A bill can be in paid/unpaid state

Analysis application objects:

The Checkout use case suggests the following additions:

- The CustomerObject is linked with a CartObject.
- A BillObject which can be paid or unpaid.
- A CartObject is associated with a BillObject.

The UML diagram from Figure 16 gives a description of the identified Objects with their methods, attributes and associations.

6.6 Developed (base) PIM

The identified application objects from the use-case analysis have led to the following PIM. This PIM is intended to fulfill the required functionality of the shopping cart and functions as the basis for which the presentation and business logic PSM's have to be constructed.

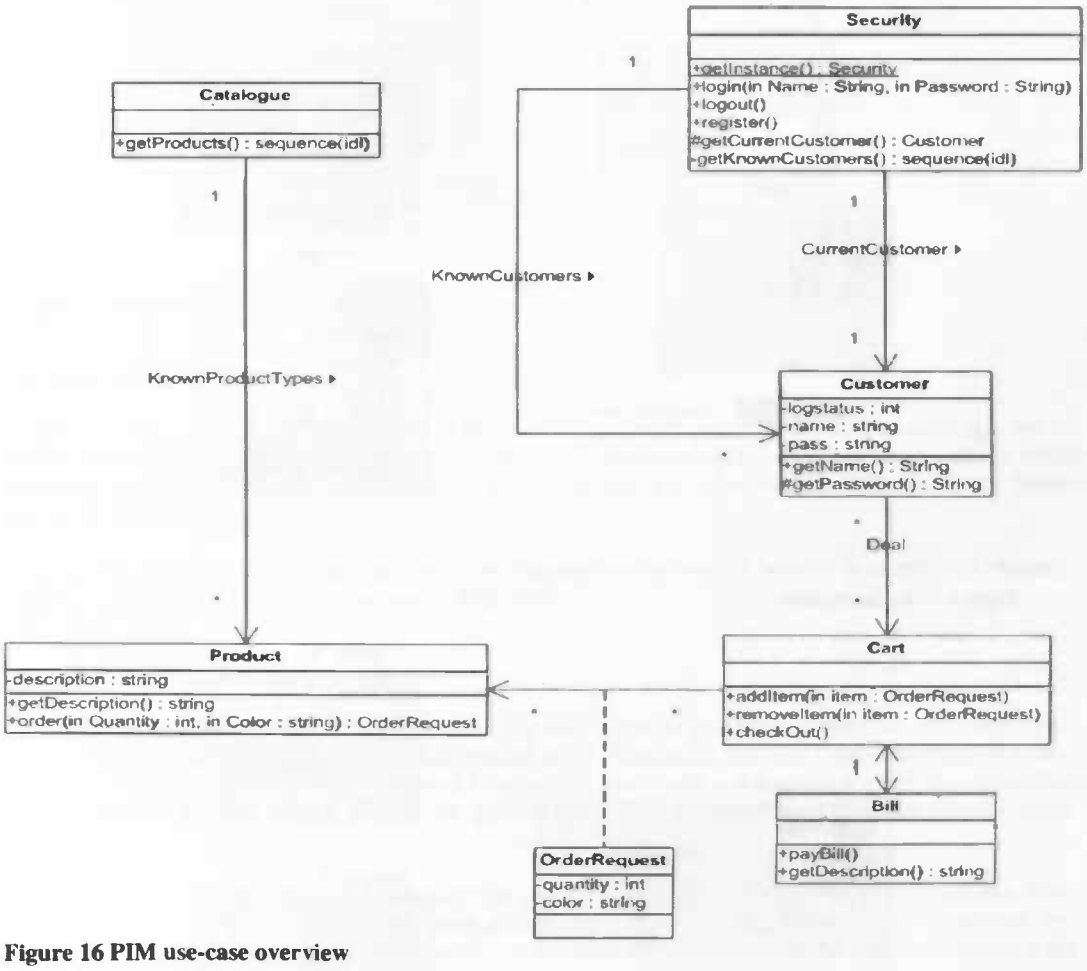

Figure 16 PIM use-case overview

6.7 Graphical representation of the use-cases

This section graphically illustrates how the application functions from the users points of view. The (web) pages are implemented through jsp files, the arrows indicate actions that are taken by the user and the shaded areas indicate information that is dependent on the application state.

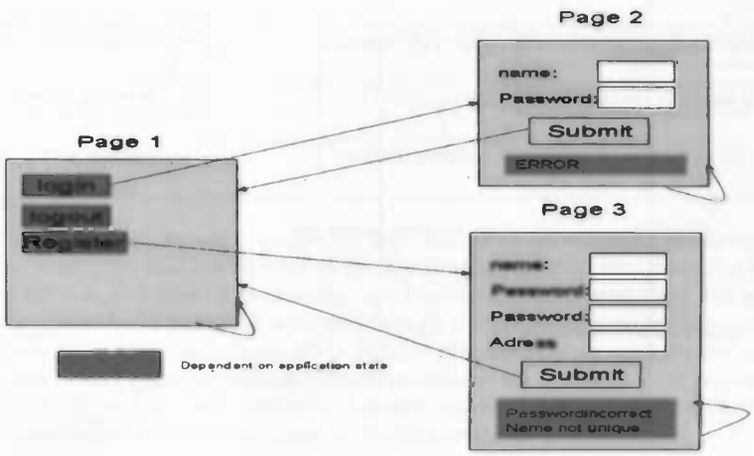

Figure 17 login/register

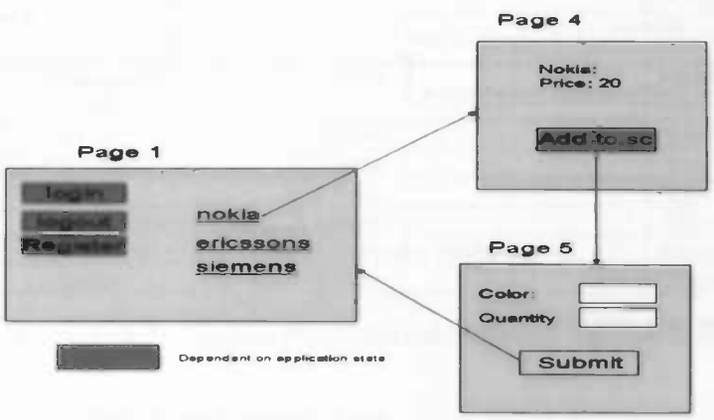

Figure 18 Acquire product → Add to shopping cart

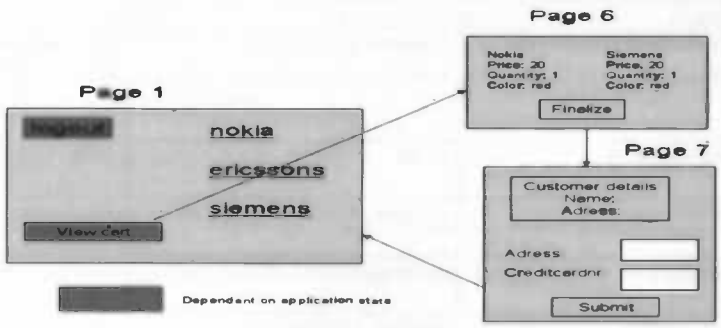

Figure 19 Acquire products → Checkout

6.8 Presentation PSM

The PIM from the previous chapter is taken as the basis (Twin Object principle) for the business logic and presentation PSM. The analysis in this section investigates if the functionality of the application can be fulfilled by adding stereotypes indicated through '<< >>' at the PSM level and ocl-statements indicated by '{}' brackets at both the PSM and PIM level.

The graphical representations of the use-cases from the previous section will be used to identify the required stereotypes and ocl-statements for the presentation PSM.

6.8.1 Identified stereotypes and ocl-statements

The graphical representation from the previous section shows that the presentation PSM not only has to take care of graphical issues like displaying a login method as a button or link, but also issues that requires some control. For example a login request (indicated by an arrow) has to start a new window (page 2) and the display of the login\logout buttons is dependent on the application state i.e. a logged customer is only shown a logout button.

The tables below illustrate the identified stereotypes and ocl-statements from the graphical illustrations. The tables are split up in three parts; the presentation stereotypes (indicate the general required UI-elements), the control stereotypes (indicate some general control issue expressed by an arrow) and pre-post conditions in the form of ocl-statements (indicated by the shaded areas in the picture).

Login/Register use-case:
The login/logout/register methods from the base PIM are all shown through buttons. Buttons are general presentation constructs and can be expressed through a <<button>> stereotype. The (general) action that has to be taken by a login and register button if pressed is that of the start of a new window (which can be used for inserting the required parameters); for this a <<start_window>> control stereotype is suggested. The logout button however has to directly invoke the logout method; for this a <<invoke_method>> is suggested. All three methods from page 1 show/hide themselves dependent on the application state; for this some pre-conditions expressed in ocl-statements can be used.

Table 1 Login/register page1

Page 1 (homepage)			
UI elements:	**UI element stereotype:**	**Control stereotype:**	**Ocl-statements: {Preconditions}**
Page 1	<<window>>	none	none
Login	<<button>>	<<start_window>>	{login-precondition}
Logout	<<button>>	<<invoke_method>>	{logout-precondition}
Register	<<button>>	<<start_window>>	{register-precondition}

The Submit button in page 2 actually invokes the login request i.e. an <<invoke_method>> stereotype is suggested. The parameters that are required by the login method are displayed through a <<text>> stereotypes and the parameters can be edited by the user through a <<inputeditor>> stereotype. The error message that is displayed after a failed login attempt is presented through a <<text>> and can easily be controlled through a {loginParameterCondition} which checks if the filled in parameters of the login method are correct and not by handling an exception of the login method itself. Note that separately checking the parameters of the login method invocation through a {precondition} is a general construct that is much simpler then handling the exception of a method invocation which puts additional constraints on this method.

Table 2 Login/register page 2

Page 2			
UI elements:	**UI element stereotype:**	**Control stereotype:**	**Ocl-statements: {Preconditions}**
Login (page2)	<<window>>	None	None
Name	<<text>>	None	None
Inputfield	<<inputeditor>>	None	None
Password	<<text>>	None	None
Inputfield	<<inputeditor>>	None	None
Submit	<<button>>	<<invoke_method>>	None
Error	<<text>>	None	{loginParameterCondition}

The register page i.e. Page 3 is similar to Page 2. The main difference is that registering requires more parameters than a login.

Table 3 Acquire product → add to shopping cart

Page 1			
UI elements:	**UI element stereotype:**	**Control stereotype:**	**Ocl-statements: {Preconditions}**
Menu (invisible)	<<list>>	None	None
Nokia	<<hyperlink>>	None	None

Ericsson	<<hyperlink>>	None	None
Siemens	<<hyperlink>>	None	None
View cart	<<button>>	<<start_window>>	{login-precondition}

Page 4 and Page 5 are not handled; the deduction of the stereotypes and ocl-statements however is straightforward and is left to the reader.

Acquire product → checkout:

Table 4 Acquire product → checkout

Page 1			
UI elements:	**UI element stereotype:**	**Control stereotype:**	**Ocl-statements: {Preconditions}**
View cart	<<button>>	<<start_window>>	{login-precondition}

Page 6 and Page 7 are not handled; the deduction of the stereotypes and ocl-statements however is straightforward and is left to the reader.

6.8.2 Presentation and business logic PSM analysis

The tables in the previous section show that methods in the presentation PSM can be associated with multiple presentation stereotypes, there is need for separate control stereotypes and that there is a mixing of control and presentation stereotypes. The business logic PSM itself is quite similar to the (base) PIM.

Presentation and control stereotypes:
For example Table 1 and Table 2 show that the login and register methods are associated with double presentation stereotypes i.e. <<button>> on page 1 and a <<window>> for page 2.
The mixing of control and presentation issues is illustrated by methods that have the same presentation stereotype but that require different control stereotypes; for example both login and logout are represented through a <<button>> but the login method requires an indirect method invocation through a <<start_window>> and logout directly invokes a method through the <<invoke_method>> construct.
Another illustration of this mixing of control and presentation is that methods with different presentation stereotypes can require the same control stereotypes; for example the logout method is represented through a <<button>> but could have been represented through a <<hyperlink>> without problems.

Another issue is the positioning of the presentation stereotypes relative to each other; for example should a list of elements be presented horizontal or vertical (see Table 3)?

OCL-statements:
The tables in the previous section also show presentational issues dependent on the application state. For example the login button must show itself if the application is in a "logged out" state. The (base) PIM and the presentation PSM lack this kind of stateful information. The presentation objects need to query the application objects for this information. The query for getting the stateful information from the application objects is a control issue that can be easily expressed in for example a {loginPrecondition} through an ocl-statement as shown in Table 1.

Business logic PSM:
The business logic PSM itself is almost equal to the (base) PIM. The business logic PSM however does have required application state information like the "logged in" state. As can be seen in the next section the business logic PSM and (base) PIM can be made identical thereby making the business logic PSM obsolete.

Identified issues:

- Double stereotyping i.e. what is the general construct?
- Mixing of control and presentation; this can easily grow out of control because there are many dependencies between them i.e. what is a simple general construct that handles this mixing?
- The relative positioning of UI-elements towards each other i.e. stereotypes are not suitable for expressing a hierarchy of UI-elements.
- Lack of information in the (base) PIM and Presentation PSM with respect to the application state.

6.9 Proposed solution

From the analysis in the previous section it showed that there are problems with fulfilling the required functionality of the application by using stereotypes and ocl-statements. The issues that were identified at the presentation PSM are; need for double stereotyping and relative positioning of UI-elements; a mixing of presentation and control issues and lack of required (stateful) information in the (base) PIM and thereby in the presentation PSM.

The proposed solution for solving the double stereotyping, the relative positioning issues of the UI-elements and the mixing of control and presentation is the usage of separate meta-types for both presentation and control issues. Meta-types are more powerful UML-constructs then stereotypes, the double stereotyping and relative positioning issues can be handled by presentation meta-types. A downside of meta-types compare to stereotypes is that they are more complex. The mentioned problems and a similar solution are also proposed in [40].

The lack of information at the (base) PIM which is required by the presentation objects can be handled by realizing part of the control in the business logic. The proposed solution for realizing this functionality consists out of two steps. First the class methods in the (base) PIM are extended by ocl-statements in the form of pre-conditions which can be derived from the application behavior. Second the (base) PIM interfaces are extended with separate methods for implementing these pre-conditions. These separate methods are now available in the presentation PSM and the generated presentation objects can now query the required stateful information from the application objects through these methods. Note that the extended PIM obsoletes the need for a separate business logic PSM i.e. the application objects can be directly generated from the extended PIM. The proposed solution by steering the presentation objects through pre- and postconditions defined on the application objects are inspired by [45].

Note: Realizing part of the control in business logic by modeling corresponding methods for the pre- and postconditions in the PIM still adheres to the Twin Object principle.

6.10 Extended PIM

In section 6.9 it was identified that the PIM can be extended with pre- and postconditions in the form of ocl-statements and corresponding methods thereby realizing part of the presentational control in the business logic. An example of how this can be done is given in the figure below; the example illustrates the identified pre-condition and the related method that can implement the precondition for enabling the presentation of the login method button. For the logout and register method a similar approach can be taken. The figure also shows the required pre-condition and inferred method for enabling the presentation of an error message by checking the parameters of the login. Note that is a general mechanism that can be applied to presentational issues which are dependent on the outcome of some method but does not require that this method is rigorously standardized.

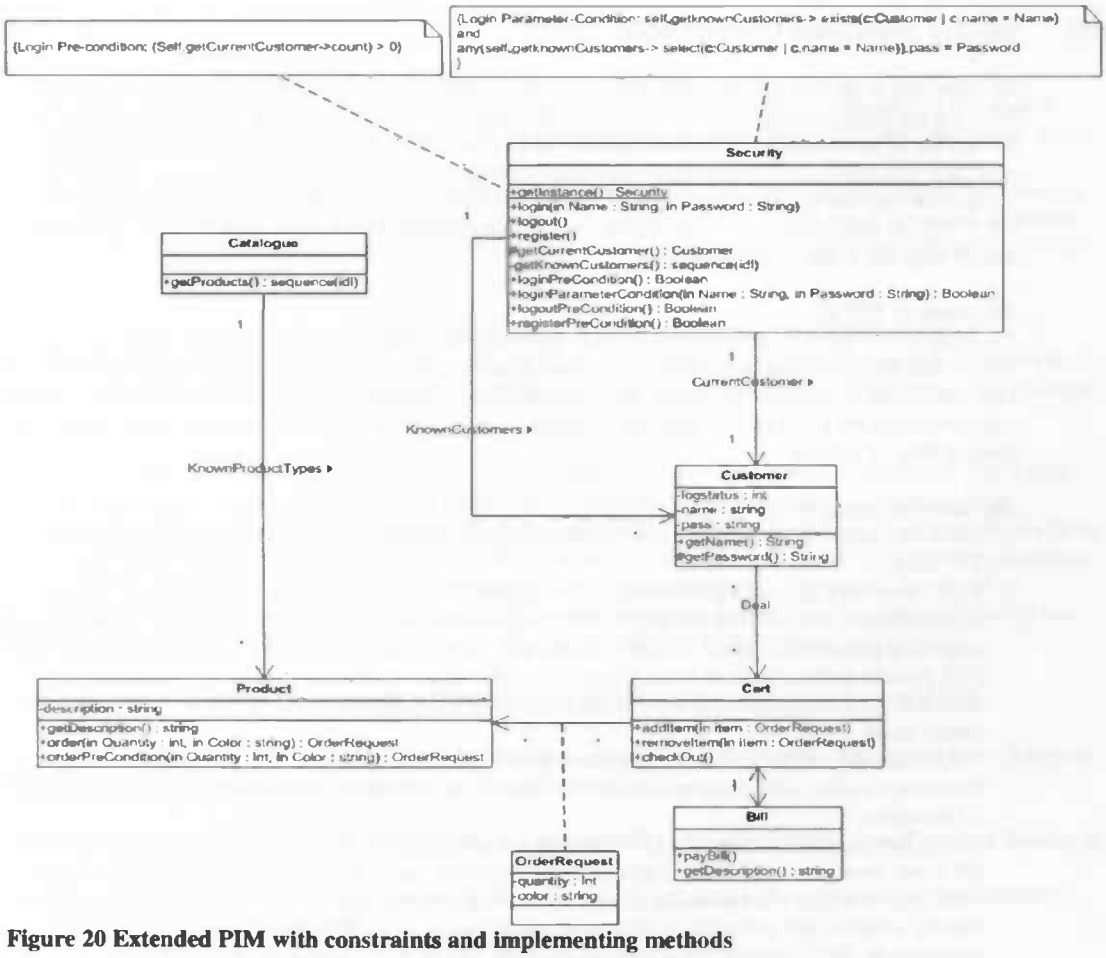

Figure 20 Extended PIM with constraints and implementing methods

6.11 Tailored Presentation Control Classes (TPCC)

As identified in section 6.8 the possibility of adding <<stereotypes>> and ocl-statements to the presentation PSM are not strong enough for a model driven development of the application. The double (multi) stereotyping, the mixing of presentation and control, and the lack of required information in the (base) PIM hinder this approach. For this reason Tailorable Presentation Control Classes TPCCs are introduced. The TPCC approach is based on one fundamental idea (observation) and is (partly) paired with the proposed solutions from section 6.9. TPCC is explained below and intended to support a model driven development of the application while maintaining the Twin Object Principle.

The basis of TPCC:
The idea of TPCC is based on the observation that there are certain standard dialogue forms; through which certain standard goals can be achieved in the user-interface. For example the preparing of a method invocation (like login) can be achieved through a standard dialogue i.e. opening a page, entering the required parameters and submitting the request. The standard dialogue form is directly related to a certain control meta-type i.e. a general form of control.

The starting points of TPCC are:
1. TPCC incorporates standard general forms of control (control meta-types) in its meta-model (basis of TPCC!!).
2. TPCC distinctly separates presentation from control. In section 6.8 it was identified that there are of lot of control related issues which are mixed with the presentation of the application, for this reason it is chosen to create a separate meta-model for presentation and control (see section 6.9). TPCC still has a link with the PIM thereby maintaining the Twin Object Principle. The PAC pattern is used for controlling the user interface. As mentioned in section 5.6 the PAC pattern supplies this clear separation of control and presentation.
3. TPCC uses meta-types instead of <<stereotypes>> for its presentation. This is based on the observation that UI-elements need to be positioned relative to each other and that a method can be associated with multiple UI elements.
4. Part of the control aspect within TPCC is realized as separate methods called control methods. In section 6.9 it was mentioned that OCL-statements like {method_name, precondition:(some_condition)} in combination with a corresponding control method can be placed in the PIM (becoming the extended PIM), thereby allowing the presentation objects to adapt their state by querying the application objects for stateful information. TPCC uses this principle for realizing part of its control in the business logic. Note the introduction of control methods in the extended PIM obsoletes the requirement for a separate business logic PSM.

Note: The user-interaction introduces new dialogues, which result in additional UI elements and new states of the application. The result of these new UI elements and new states is that TPCC still adheres to the Twin Object principle but not all properties are respected i.e. TPCC can not be kept reflexive in its 1:1 mapping between UI elements and business logic.

6.12 PoC and TPCC

For the further development of the PoC we are going to test the 4 starting points of TPCC through a control analysis. The control analysis investigates some use-case examples and is intended to develop the required meta-models and corresponding models required for fulfilling the applications control and presentation issues. The control analysis is paired with an investigation of the required additions to the business logic i.e. control methods. The steps that are taken in the control analysis are explained below.

Step 1 — What are the general forms of control and presentation?

- Identifying the general forms of control as exhibited by the selected example i.e. identifying the control meta-type(s). Figure 23 (Control meta-model) can be consulted for the complete set of identified control meta-types.
- Identifying the possible presentations of these general control forms i.e. identifying the linked presentation meta-types for the identified control meta-types. Figure 22 can be consulted for the complete set of identified presentation meta-types (Presentation meta-model). The link between both meta-models completes the TPCC meta-model (see Figure 24).

Step 1.1 — What are the corresponding general forms of control located in the business logic?

- Identifying the behavior of the application objects that is required for supporting the identified control meta-type i.e. identifying the type of control method located in the application object for supporting the control meta-type(s).

Once steps 1 and 1.1 have been completed for all the examples, the TPCC meta-model (skeleton) (see Figure 24) can be developed and instantiated to the Control and Presentation model skeleton (see Figure 25).
The identified general control forms in the TPCC meta-model (skeleton) still have to be made specific for being able to fulfill the functionality of the application. For this reason the linked control and presentation meta-types from the TPCC meta-model will be instantiated to *presentation control classes* from the Control and Presentation model and an analysis will follow on how to the identified general forms of control (control meta-types) linked with their possible presentations can be further extended with attributes and relations.

Step 2 — How to model a specific application from the identified general forms of control and presentation?

- Identifying the required specialization of the identified general control, presentation forms i.e. modeling the specific control and presentation meta-type attributes and relations.

Step 2.1 — How to model a specific application for the identified corresponding form of control located in the business logic?

- Identifying the required specialization of the business logic i.e. modeling the required control method(s) in the application class (extended PIM).

Once steps 2 and 2.1 have been completed for all the examples the complete TPCC meta-model (see Figure 24) can be developed. The TPCC meta-model can be instantiated to the Control and Presentation model without the specific attribute-values and relation-values required by the application. For this reason the *presentation control classes* will be investigated on missing attribute/relation values.

Step 3 — What are the attribute/relation values of the presentation control classes in the Control and Presentation model?

- Identify and fill in the missing values of the attributes/relations from the presentation control classes.

Step 3.1 — What are the name(s) of the control method(s)?

- Identify and fill in the name(s) of the corresponding control methods in the PIM (thereby becoming the extended PIM).

Once steps 3 and 3.1 have been done for all the examples the complete Control and Presentation model (see Figure 25) and the complete extended PIM (see Figure 21) can be constructed. We still have to generate the code from these models to the jsp-files (a.k.a the *tailored presentation control classes*) for this we assume a template called a *tailorable presentation control class* which uses the Control and Presentation model to fill in its variable parts thereby making it specific for the use-case examples. See section 6.14 for a source code analysis.

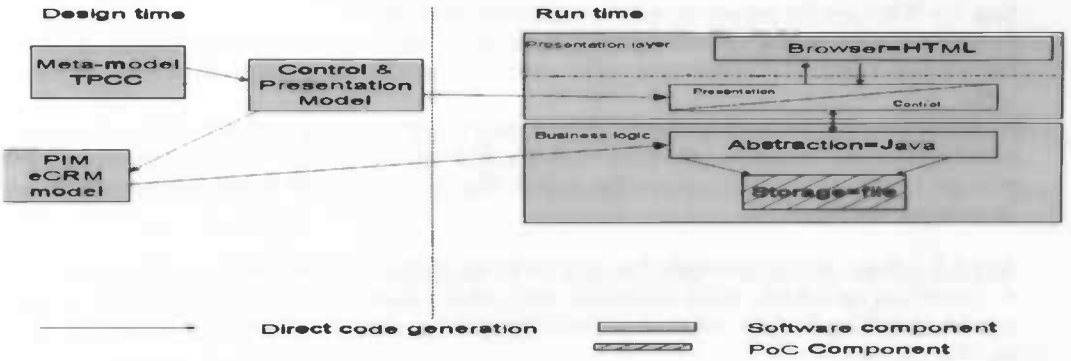

Figure 21 PoC architecture using TPCC

The remainder of this chapter is organized as follows: First a control analysis for some relevant use-cases is conducted; this control analysis is intended to develop the TPCC meta-model (see Figure 22, Figure 23, Figure 24) and required models (see Figure 25 and Figure 20) from which the jsp and java code has to be generated. Secondly the required jsp code has to be implemented and analyzed (see section 6.14); this analysis is intended to show that the code indeed could have been generated from the created Control and Presentation model. Thirdly a literature study is done to determine the place of TPCC with respect to other developing techniques. This chapter is concluded with an evaluation of the Proof of Concept.

6.13 Deriving meta-models for TPCC through a Control Analysis

For the control analysis the login/register use-case and the acquire product use-case will be investigated, the following examples have been selected for a thorough examination:

1. Example 1: (page 1) the login-button is only allowed when a user is not already logged-on in the current session; otherwise the logout button is shown. The action taken by the login button is the start of a dialogue which can be used to enter parameters. The action taken by the logout button is a direct method invocation on the security object.
2. Example 2: (page 2) the submit request with a wrong name/password combination is deemed incorrect i.e. a text field with an error message is shown.
3. Example 3: (page 1) the user can order those products that are known to the catalogue object.

6.13.1 Login/Register use-case (example 1)

Example page 1: log-in is only allowed when a user is not already logged-on in the current session.

Step 1 – What are the general forms of control and presentation?

- Identifying the general forms of control as exhibited by the selected examples i.e. identifying the control meta-type(s)
- Identifying the possible presentations of this general control form i.e. identifying the linked presentation meta-types for the identified control meta-type.

Login + Register method:

1. The Control meta-type for the login/register method is that of a "Create Method Request" with a relation to another dialogue that will be started in which the parameters of the method will be entered and the method invoked.
2. The Presentation meta-type for the login + register method is that of a "Button".

Logout method:

1. The Control meta-type for the logout method is that of a "Do Method Request". When selected, a direct method invocation on an application object will follow.
2. The Presentation meta-type is that of a "Button".

General presentation (home page):

1. The Control meta-type for the start of some home page is that of a "General Dialogue" which can hold a collection of control meta-types.
2. The Presentation meta-type for a "General Dialogue" is that of a "Window".

Step 1.1 – What are the corresponding general forms of control located in the business logic?

- Identifying the behavior of the application objects that is required for supporting the identified control meta-type i.e. identifying the type of control method located in the application object for supporting the control meta-type(s).

1. The added application behavior is that of control methods that evaluate the pre-conditions for "normal" methods like login, register and logout.
2. The "General Dialogue" does not require any pre-conditions.

Step 2 – How to model a specific application from the identified general forms of control and presentation?

- Identifying the required specialization of the identified general control, presentation forms i.e. modeling the specific control attributes and relations.

The "Create Method Request" extended by:

- Its "Name" attribute.
- Its relation with a View Object, e.g. in terms of a button.
- Its relation to the dialogue that will be used to enter parameters and submit the request (see Example 2).
- Its "applicationMethodName" attribute and "applicationClassName" attribute for which the invocation is requested. From the "applicationMethodName" attribute the name of the specific pre-condition control method will be inferred.

The "Do Method Request" is extended by:

- Its "Name" attribute.
- Its relation with a View Object, e.g. in terms of a button.
- Its "applicationMethodName" attribute and "applicationClassName" attribute for which the invocation is requested. From the "applicationMethodName" attribute the name of the specific pre-condition control method will be inferred.

The "General Dialogue" is extended by:
- Its "Name" attribute.
- Its relation with a View Object, e.g. in terms of a window.
- Its relation to other dialogues.

Step 2.1 — How to model a specific application for the identified corresponding form of control located in the business logic?
- Identifying the required specialization of the business logic i.e. modeling the required control method(s) in the application class (extended PIM).

1. The application class in the PIM is extended by modeling "normal" control methods that evaluate preconditions.

Motivation general constructs:
The observation is that in various applications mini dialogues will occur (e.g. presented as buttons) that do not directly invoke a method but that invoke a dialogue for setting the parameters of a method. Thus, a generic construct such as this is justified.
A separate general construct is provided for directly invoking a method, this is called: "Do Method Request" (see Figure 23).
A "General Dialogue" is needed as a general starting point and for holding a collection of dialogues.
Control methods for evaluating the pre-condition of a "normal" method is a general construct that can be used by presentation objects to check whether a method will indeed be offered to the user.

Step 3 — What are the attribute/relation values of the presentation control classes in the Control and Presentation model?
- Identify and fill in the missing values of the attributes/relations in the Control and Presentation model.

1. The "name" attribute of the "Create Method Request" control meta-type is "login Create Method Request", this presentation control class refers to the "login Prepare Method Request Dialogue" (see example 2), the method name is "login" as defined on the "Security" class.
2. The name of the "Do Method Request" is "logout Do Method Request", the method name is "logout" as defined on the "Security" class.
3. The "name" attribute of the "General Dialogue" is "eCRM Application General Dialogue", this presentation control class refers to both the "loginCreateMethodRequest" and "logoutDoMethodRequest presentation" control classes.

Step 3.1 — What are the name(s) of the control method(s)?
- Identify the name of the control method(s) in the PIM (thereby becoming the extended PIM).

1. The names of the control methods will become loginPreCondition(), logoutPreConditon() and registerPreCondtion(). The implementation of the actual functionality can be done from the corresponding ocl-statements (preconditions). This is illustrated in Figure 20 for the login method, for the logout and register method this is similar.

6.13.2 Login/Register use-case (example 2)

Example page 2: log-in with a wrong name – password combination is deemed incorrect.

Step 1 — What are the general forms of control and presentation?

- Identifying the general forms of control as exhibited by the selected examples i.e. identifying the control meta-type(s)
- Identifying the possible presentations of this general control form i.e. identifying the linked presentation meta-types for the identified control meta-type.

1. The Control meta-types for the login/register method invocation are that of a "Prepare Method Request Dialogue" and "Method Parameter Editor". These dialogues will offer the behavior to enter the parameters and submit the request. When the request is submitted it firstly checks the correctness of the parameters of the request. If the parameters are incorrect, an error message is presented, and the dialogue will remain active.
2. The Presentation meta-type for the login method invocation is that of a "Composite View" (could be a "Window"). The presentation meta-types for showing the name of the required parameters and showing the inputfield are "Text" and "InputEditor". The presentation meta-type for showing the error message is "Text".

Step 1.1 — What are the corresponding general forms of control located in the business logic?

- Identifying the behavior of the application objects that is required for supporting the identified control meta-type i.e. identifying the type of control method located in the application object for supporting the control meta-type(s).

1. The added application object behavior is that of a control method type that checks the parameters of a method invocation. The same application object also contains the method itself.

Step 2 — How to model a specific application from the identified general forms of control and presentation?

- Identifying the required specialization of the identified general control, presentation forms i.e. modeling the control meta-type attributes and relations.

The "Prepare Method Request Dialogue" is extended by:

- Its "Name" attribute.
- Its relation to the dialogues (Method Parameter Editors) which are used for controlling the settings of the parameters of a certain method.
- Its relation with the View Objects, e.g. the button used for invoking the method, the text message for representing a failed login attempt and a "Window" for holding multiple elements.
- Its "applicationMethodName" attribute and "applicationClassName" attribute for which the invocation is requested.

The "Method Parameter Editor" is extended by:

- Its relation with the View Objects in terms of "Text" and "InputEditor".
- Its "applicationMethodName" attribute, "applicationClassName" and "applicationParameterName" attribute.

Step 2.1 — How to model a specific application for the identified corresponding form of control located in the business logic?

- Identifying the required specialization of the business logic i.e. modeling the required control method(s) in the application class (extended PIM).

1. The application class in the PIM is extended by modeling a "Parameter-Condition" control method. This method will evaluate the correctness of the (combination of) parameter values.

Motivation:
For checking the correctness of parameters another generic construct is possible as well, namely where an exception of the method invocation itself is handled. This however requires that these method implementations are rigorously standardized; the use of a separate generated control method does not require that.
Last but not least a generic construct for a login dialogue could be useful construct as well; this has not been chosen here for instructional reasons.

Step 3 — What are the attribute/relation values of the presentation control classes in the Control and Presentation model?
- Identify and fill in the missing values of the attributes/relations in the Control and Presentation model.

1. The name attribute of the "PrepareMethodRequestDialogue" is loginPrepareMethodRequestDialogue", this presentation control class refers to the "nameMethodParameterEditor" and "passwordMethodParameterEditor" presentation control classes.
2. The error message is given as a text: "Error, incorrect login", note that this error message is linked to the loginPrepareMethodRequestDialogue.
3. The method name and class name attributes are "login" and "Security".

Step 3.1 — What are the name(s) of the control method(s)?
- Identify the name of the control method(s) in the PIM (thereby becoming the extended PIM).

1. The name of the control method will become "loginParameterCondition(some_condition)". The implementation of the actual functionality can be done from the corresponding precondition. This is illustrated in Figure 20.

6.13.3 Acquire Product use-case (example 3)

Example 3: The user can order those products that are known to the catalogue object.

Step 1 — What are the general forms of control and presentation?
- Identifying the general forms of control as exhibited by the selected examples i.e. identifying the control meta-type(s)
- Identifying the possible presentations of this general control form i.e. identifying the linked presentation meta-types for the identified control meta-type.

1. The Control meta-types are that of a "List Elements" and "Object Previewer". The "List Elements" provides the functionality to present elements from a list and select from those elements; for this it uses the "Object Previewer" control meta-type which provides the functionality to show preview information about an object from the list.
2. The Presentation meta-types for representing a "List Elements" is that of a "RepeatedCluster", the elements "Object Previewer" in the list are represented through an atomic "Hyperlink" object.

Step 1.1 — What are the corresponding general forms of control located in the business logic?
- Identifying the behavior of the application objects that is required for supporting the identified control meta-type i.e. identifying the type of control method located in the application object for supporting the control meta-type(s).

1. The application object has a method that returns a sequence of objects (already present in the CatalogueObject).

Step 2 – How to model a specific application from the identified general forms of control and presentation?

- Identifying the required specialization of the identified general control, presentation forms i.e. modeling the control meta-type attributes and relations.

The "List Elements" is extended by:

1. The relation to the presentation control class (Object Previewer) that is used to select any of the elements
2. The "applicationClassName"attribute and "applicationMethodName" attribute for invoking the method to get a sequence of elements.
3. The presentation of the list in terms of a "RepeatedCluster".

The "Object Previewer" is extended by:

1. The View Object in terms of a "Hyperlink".
2. The "applicationClassName"attribute and "applicationMethodExpression" to get the preview information of some specific element.

Step 2.1 – How to model a specific application for the identified corresponding form of control located in the business logic?

- Identifying the required specialization of the business logic i.e. modeling the required control method(s) in the application class (extended PIM).

1. No further need.

Motivation:
The "List Elements" seems like an obvious construct. Note that this construct is one where output (of the method getProducts, see below) is linked to potential input, i.e. using the "Object Previewer" to navigate to the dialogue to present the product i.e. "PresentObjectDialogue" (not handled).

Step 3 – What are the attribute/relation values of the presentation control classes in the Control and Presentation model?

- Identify and fill in the missing values of the attributes/relations in the Control and Presentation model.

The name attribute of "List Elements" is "productsListElements" and it uses the "productObjectPreviewer" presentation control class to select any of the elements in the list. The "productListElements" presentation control class is tailored with the method name "getProducts", as defined on the "Catalogue" class.
The "productObject Previewer" presentation control class uses the "getDescription()" method, with an associated hyperlink presentation.

Step 3.1 – What are the name(s) of the control method(s)?

- Identify the name of the control method(s) in the PIM (thereby becoming the extended PIM).

1. No further need.

6.13.4 Meta-models

The UML models from this section are derived from the control analysis. Figure 22 the Presentation (UI) meta-model represents the identified presentation meta-types. Figure 23 presents the Control meta-model from the identified control meta-types. Figure 24 represents the complete TPCC meta-model i.e. the Control meta-model, Presentation meta-model and their link. Figure 25 presents the Control and Presentation model i.e. the instantiated TPCC meta-model that is required for fulfilling the functionality and presentation of the use-case examples.

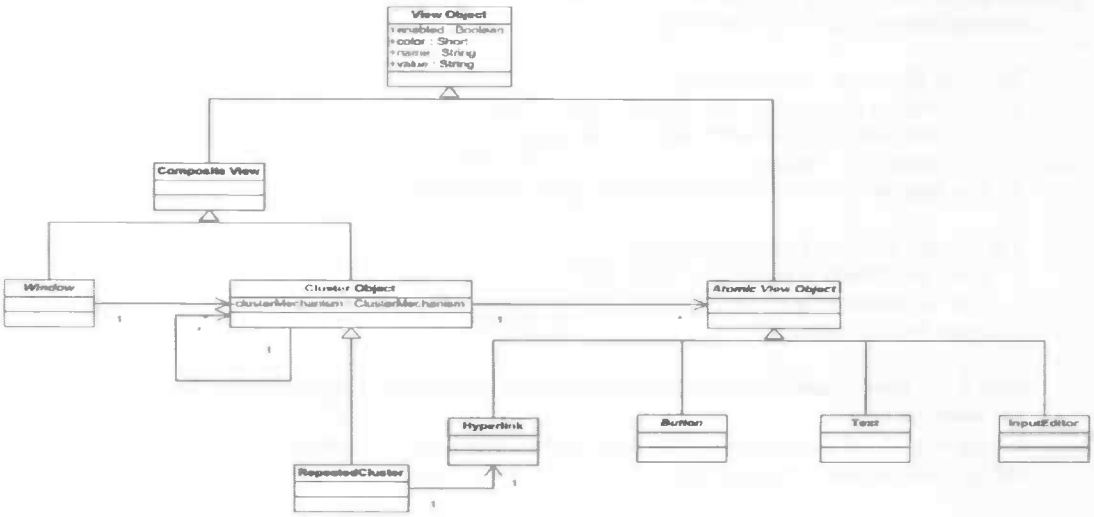

Figure 22 Presentation meta-model

The Presentation meta-model that can represent the hierarchy of UI elements is illustrated above; this Meta model takes into account the double stereotyping and the relative positioning of UI-elements. In the hierarchy of UI (View) elements Cluster Objects play a prominent role. This mechanism has been taken over from ET++[47] and can be used to create a tabular layout of the different UI elements.

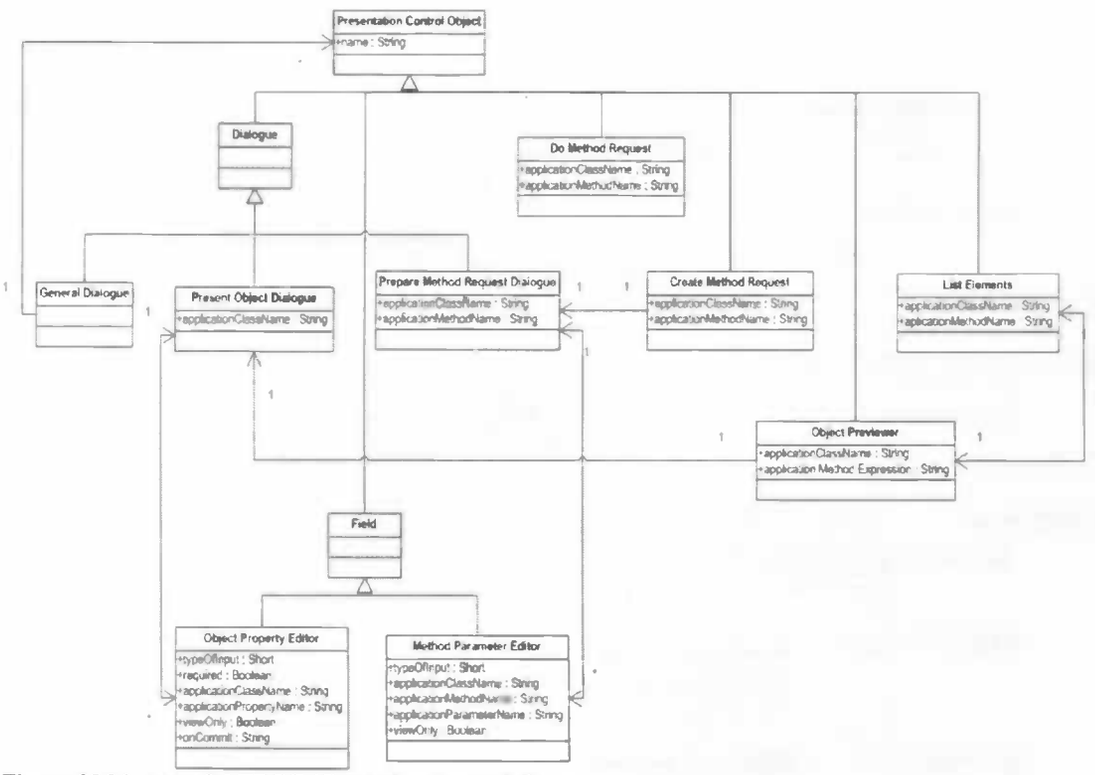

Figure 23 Meta model TPCC (Control meta-model)

The TPCC meta-model (Control meta-model) constructed from the identified control meta-types describe those tailorable control classes as were relevant for the example. Properties such as "applicationClassName", "applicationMethodName" and "applicationParameterName" are used to tailor a presentation control class to link to certain application classes and methods.

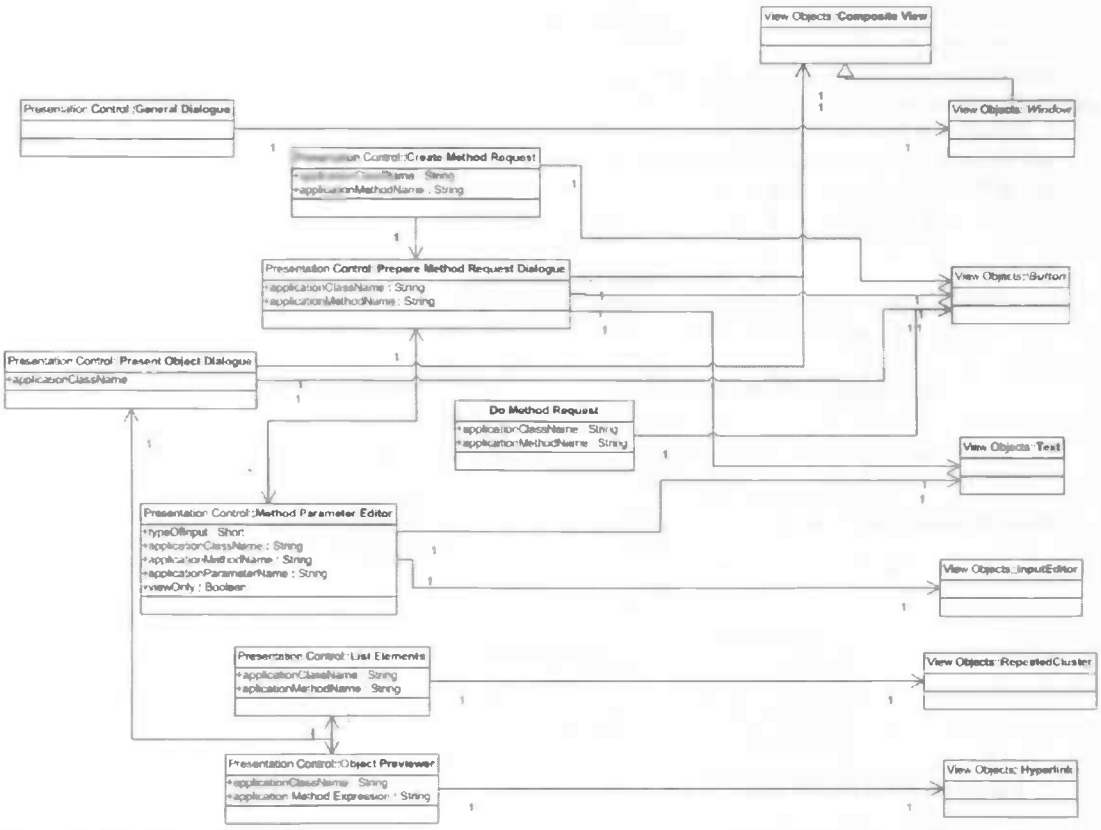

Figure 24 TPCC meta-model

Figure 24 shows the third part of the TPCC meta-model. This figure describes how the Control meta-types can be tailored by means of View Objects i.e. it shows the possible presentations of the control meta-types. The figure also shows the strict decoupling between the application and the pure presentation.

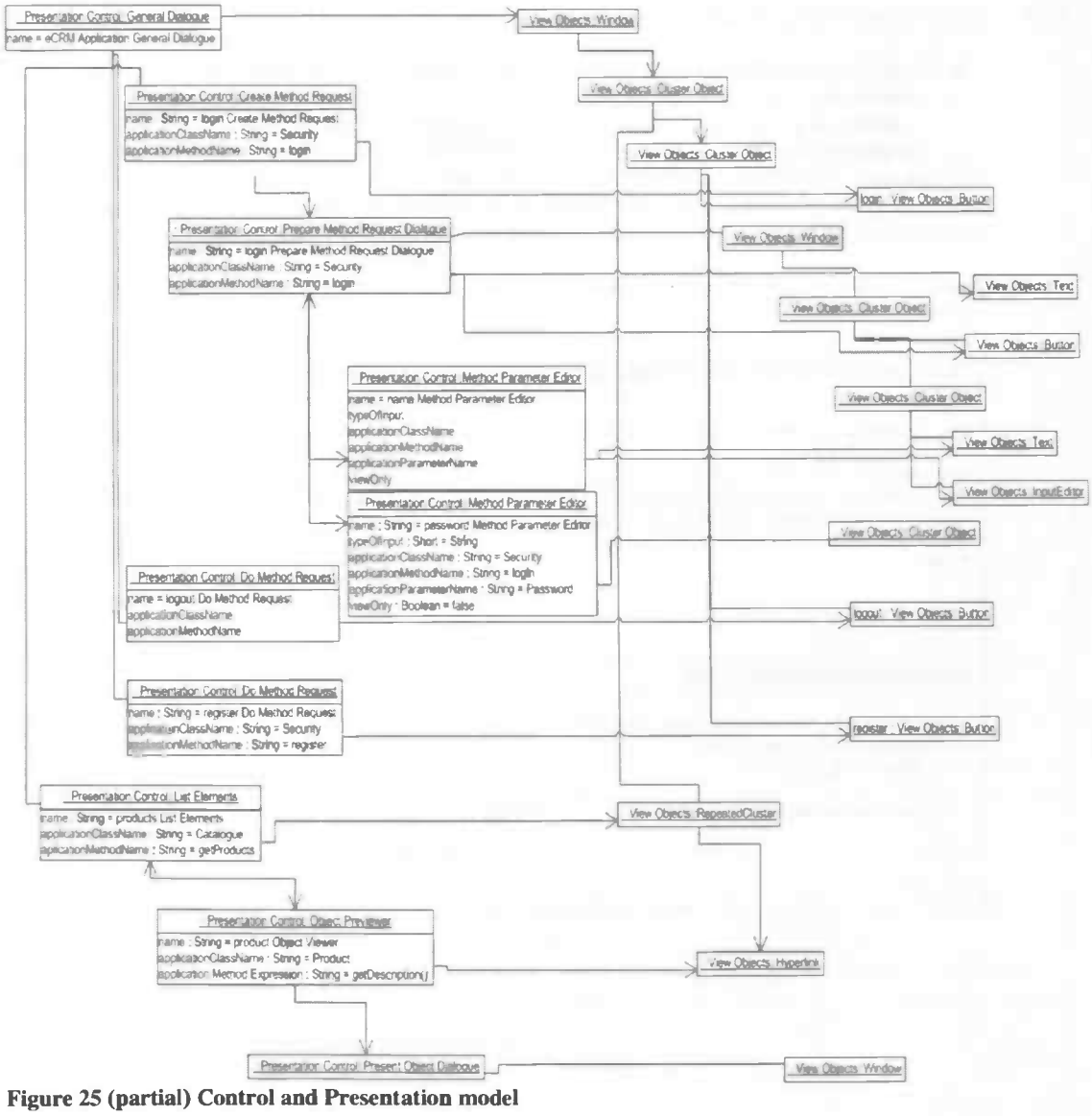

Figure 25 (partial) Control and Presentation model

The Control and Presentation model from Figure 25 presents the instantiated TPCC meta-model, and thus the model that is used to tailor the presentation control classes for the example application. A double hierarchy is formed: The hierarchy of the models that describe the presentation control classes and the hierarchy that describes the view elements. View elements parameterize the model of the presentation control classes.

Note: A mistake has been made in the model from Figure 25 where a register method is represented through a doMethodRequest, this should be CreateMethodReques with associated presentation control classes as is done for loginCreateMethodRequest.

6.14 Source code implementation for the login use case

The important pieces of source code (jsp's) that were required for implementing the presentation of the use-case examples from the control analysis in section 6.13 are illustrated in the table below. The source code in the form of jsp files is going to be compared to the Control and Presentation model as illustrated in Figure 25. In the jsp implementation both the control aspect and the presentation aspect have been integrated into one file. The analysis assumes basic templates (tailorable presentation control class) that are present for handling the static part of the presentation; the variable parts of the template are illustrated by an underline and should be available in the Control and Presentation model for an automatic code generation.

Note: The required java code for the implementation of the business logic (extended PIM) can be directly derived from Figure 20.

eCRMapplicationGeneralDialogue.jsp

```
<html>
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="page"/>
<title>ecrmapplicationgeneraldialogue</title>
<jsp:include page="loginCreateMethodRequest.jsp"/>
<jsp:include page="logoutDoMethodRequest.jsp"/>
<jsp:include page="registerCreateMethodRequest.jsp"/>
<jsp:include page="productsListElements.jsp"/>
</html>
```

loginCreateMethodRequest.jsp

```
<html>
<jsp:useBean id="so" class="ecrm.SecurityObject"
scope="session"/>
<% if (so.loginPreCondition()){%>
<form action="loginPrepareMethodRequestDialogue.jsp">
   <p>
     <input type=submit value=Login>
   </p>
</form>
<% } %>
</html>
```

logoutDoMethodRequest.jsp

```
<html>
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="session"/>
<% if (so.logoutPreCondition()){%>
<form action="logout.jsp">
   <p>
     <input type=submit value=Logout>
   </p>
</form>
<% } %>
</html>
```

loginPrepareMethodRequestDialogue.jsp

```
<html>
<%@ page import="ecrm.*" %>
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="session"/>
<jsp:useBean id="customer" class="ecrm.Customer" scope="page"/>
<jsp:setProperty name="customer" property="*"/>

<title>loginpreparemethodrequest.jsp</title>

<form action="loginPrepareMethodRequestDialogue.jsp" method="get">
   <p>
   Name:    <input type="text" name=name>
   </p><p>
  Password:<input type="text" name=password>
   </p><p>
   <input type=submit value=Submit>
   </p>
</form>

<%
      if (customer.getName()!= null){%>
<%
            if ((boolean)so.loginParameterCondition(customer.getName(), cust
omer.getPassword())){
                  so.Login(customer.getName(),customer.getPassword());%>
                  <jsp:forward page="eCRMapplicationGeneralDialogue.jsp"/>
<%
            }else{%>
                  <jsp:include page="error.jsp"/>
<%          }
      }
%>
</html>
```

registerCreateMethodRequest.jsp

```
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="session"/>
<% if (so.registerPreCondition()){%>
<form action="registerPrepareMethodRequestDialogue.jsp">
   <p>
     <input type=submit value=Register>
   </p>
</form>
<% } %>
```

productsListElements.jsp

```
<html>
<jsp:useBean id="co" class="ecrm.CatalogueObject" scope="page"/>

<title>productsListElements</title>
<br>Productlist:
<ol>
<%
      for (int i=0; i<co.getVectorLength(); i++) {
%>
<jsp:include flush="true" page="productObjectPreviewer.jsp">
 <jsp:param name="id" value="<%= i %>"/>
</jsp:include>
<%
      }
%>
</ol>
</html>
```

productObjectPreviewer.jsp

```
<%@ page import="java.util.*" %>


<jsp:useBean id="co" class="ecrm.CatalogueObject" scope="page"/>
<jsp:useBean id="po" class="ecrm.Product" scope="page"/>

<%
    String idstring = request.getParameter("id");
    int id = Integer.parseInt(idstring);
    po = co.getProduct(id);
%>
<li> <A HREF="productPresentObjectDialogue.jsp?id=<%= id %>"><%= po.getDescription()%></A>
```

productPresentObjectDialogue.jsp

```
<%@ page import="java.util.*" %>
<%@ page import="ecrm.*" %>
<jsp:useBean id="co" class="ecrm.CatalogueObject" scope="page"/>
<jsp:useBean id="product" class="ecrm.Product" scope="page"/>
<title>productPresentObjectDialogue.jsp</title>
<%
    String idstring = request.getParameter("id");
    int id = Integer.parseInt(idstring);
    Product p1 = new Product();
    p1 = co.getProduct(id);
    product = p1;
%>
description: <%= product.getDescription() %>
<form action="orderPrepareOrderRequestDialogue.jsp">
  <p>
    <input type=submit value=Order>
  </p>
</form>
```

Analysis source code:
The illustrated source code shows a direct 1:1 relation with Figure 25. For every presentation control class there is a separate jsp file. For example the filled in presentation control class "name attributes" from Figure 25 all have a direct equivalent in the form of a .jsp-file with the same name i.e. **eCRMApplicationGeneralDialogue**, **loginCreateMehtodRequest**, **loginPrepareMethodRequest**, **logoutDoMethodRequest**, **registerDoMethodRequest**, **productsListsElements**, **productObjectPreviewer** and **productPresentObjectDialogue.**

Another important observation is that the relations between model elements are directly reflected through a link in the .jsp's as well. The link in the presentation control classes of for example loginCreateMethodRequest and the loginPrepareMethodRequestDialogue in the Control and Presentation model is directly reflected through the <form action="loginPrepareMethodRequestDialogue.jsp"> tag in the corresponding jsp file. The direct link between the productListElements and the productObjectPreviewer is represented through a <jsp:include flush="true" page="productObjectPreviewer.jsp"> tag which itself has a link to the productPresentObjectDialogue which is represented through the <A HREF="productPresentObjectDialogue.jsp?id=<%= id %>"><%= po.getDescription()%></A> tag.

The source code clearly shows the link between the control classes and its link with the application objects (direct manipulation). Every separate page is aligned with object identifiers such that for every page its clear to which object or objects it is associated; this alignment is indicated through the <jsp:usebean …> tag.

The source code also shows the use of control methods that are required to steer the page, note that the names of the used control methods (see Figure 20) are present in the extended PIM and can be used. The

implementation is done through scriptlets and are indicated through the <% some-object.someCondition(); %> tag construction.

The UI elements are embedded together with control in the jsp pages; the source code therefore does not directly show the strict decoupling between the presentation and control. This strict decoupling however is present and the implementation of the UI elements show that these only use general constructs. In the table below the meaning of the jsp related UI-element code is illustrated.

View Object::Window → <html></html>
View Object::Button → <input type=submit value=?>
View Object::InputEditor → <input type=text value=?>
View Object::Text → some text

Note that not all aspects of the jsp implementation have been taken into account. For example the use of clustering objects as shown in Figure 22 and Figure 25 has not been applied. The identification/selection of a specific product is done through a workaround by scriptlets that temporary stores data. The error message which is inserted after an incorrect login has also been solved with a work around that is required for handling the first time a page is accessed.
The import thing to notice here is that clustering objects, the storage of temporary data and the first time a page is accessed are also general constructs that could have been incorporated into the TPCC meta-models.

Another aspect that has not been taken into account by the implementation is that the Control and Presentation model classes "nameMethodParameterEditor" and "passwordMethodParameterEditor" do not have a separate jsp file. In the implementation it was chosen to directly implement these basic/obvious classes that require only one line of code into the "prepareMethodRequestDialogue.jsp", but in principle these classes could/should have been implemented through a <jsp:include page="nameMethodParameterEditor.jsp"> construction.

6.15 TPCC and the Literature

A large body of work exists on model-driven or model-based user interface development (see e.g. [41]). Currently, a lot of work on model-driven (sometimes called model-based) user interface development is done in two rather separate communities: Communities that use models for web application development [40][42][43], and communities that use model-based user interface development on basis of task modeling [41][44][14] as a mechanism to enable porting user interfaces to different devices, such as hand-helds, PDAs etc. In both areas, modeling of a middle layer between data (or application) and presentation is standard practice, however, these modeling forms are different than the one presented here.
In various approaches to developing web applications the middle area between data and presentation is based on the description of mechanisms for navigation and (webpage) composition [40][42][43]. For example [40] use a Composer, a Value displayer, a Collection displayer, a Linker and a Form. Fraternali and Paolini e.g. [42] use a wide range of "navigation modes". All of these are much more than this approach concentrated on powerful mechanisms for accessing large data and web page structures. However, even while [40] describe that they have a "model-aware" action language; the strength of the approach presented here is the direct link it creates between what is possible in the user interface (with respect to operations) and the state of the application, thus providing mechanisms that are better suited to direct manipulation of application objects through methods.
Task modeling is another mechanism for modeling user interfaces [41][44]. A tasks modeling language such as CTTE [41] describes sequences of tasks, parallel sequences of tasks, alternatives (choice) between tasks, etc. Task modeling is clearly more oriented towards manipulation than the previous mentioned forms of modeling. Still, the idea of directly manipulating application objects seems far away in these approaches.
An approach that was one of the inspirations for the presented idea is that of UIDE, where pre- and postconditions defined on application objects are used to steer the user interface objects [45]. However, the decoupling between user interface and application objects is not sufficiently guarded and higher level

presentation control abstractions seem missing. More recently, a very interesting approach is that of "Naked objects" [46] where business objects are directly presented on the screen by means of reflection. The disadvantage here seems that the presentation cannot be adapted, it is almost too direct. The TPCC approach seeks a middle road with respect to simple modeling while retaining the link with direct manipulation.

6.16 Evaluation PoC

The development of the first part of the PoC shows that the functionality required by the application cannot be fulfilled at the presentation PSM by using stereotypes and ocl-statements. The double stereotyping, mixing of control and presentation issues and the lack of stateful information at the (base) PIM hinder this approach. The lack of required stateful information at the (base) PIM however can be solved by extending this PIM with preconditions in the form of ocl-statements and modeling corresponding (control) methods. Note that the required application objects can be directly generated from the extended PIM; this makes a separate business logic PSM obsolete.

For solving the problems mentioned above this thesis introduces Tailorable Presentation Control Classes (TPCCs). TPCCs offer standard forms of interaction (control meta-types) between the user and the application objects which can be formalized in simple models.
The control analysis of some use-cases show that it is indeed possible to identify general forms of control with their possible presentations (presentation meta-types) and that the required corresponding control in the business logic can be realized by modeling control methods. Furthermore the developed meta-models and models show the strict decoupling between the UI and application objects and that the application can be model driven tailored to directly link between presentation and the application objects.

Although not all aspects of the application have been taken into account in the jsp implementation, the source code analysis shows the direct 1:1 relation between the instantiated TPCC meta-model (Control and Presentation model) that is required for fulfilling the use cases and the developed source code in the form of jsp-files. The relation between the different presentation control classes in this Control and Presentation model is also directly reflected in the source code.

7 PoC and Device Flexibility using one View

In this chapter the TPCC approach from the PoC will be analyzed to see if it can fulfill the device flexibility requirement [NFR02]. In section 5.7 two ways were identified for fulfilling device flexibility [NFR02] i.e. one general view and a multiple view approach. This analysis will focus on a one general view approach using WALL. The approach will consist out of a source code inspection of relevant use-cases from the PoC.

The goal of the analysis is to see if one general view using WALL can be used to abstract away from the device flexibility thereby allowing this variability to be captured in general constructs which can be incorporated into the template used by the Tailorable Presentation Control Class.

The source code inspection investigates the differences between the implementation of some use-cases from the PoC and its implementation in WALL. The source code analysis will investigate if the WALL implementation of the presentation logic can be used for different devices i.e. will the page be rendered for different devices and will it function as intended. The analysis will also investigate if WALL indeed uses general constructs that can be incorporated into the TPCC approach.

Note: An actual implementation of the PoC through WALL has not been done. The used implementation has been derived from the examples presented in the WALL tutorial, which can be found in [15].

7.1 General WALL constructs and conventions

WALL and the related wireless markup languages WML-1.x, XHTML-MP and cHTML have different conventions and constructs than HTML. These differences are expressed in for example well-formedness rules, available tag elements and general constructs. This section gives a description of these general conventions and constructs that are used for all the examples.

Available WALL tags:
The list below sums up all the available WALL tags which are intended to overcome the differences that exist between the different markups. As mentioned before the intersection of the markups is not empty and basic constructs that are part of this intersection usually[12] do not require any specific WALL construct. It is expected that tag elements <p> belong to this intersection and have no side-effects.

a, alternate_img, b, block, body, br, caller, cell, cool_menu, cool_menu_css, document, font, form, h1,h2,h3,h4,h5,h6, head, hr, I, img, input, load_capabilities, marquee, menu, menu_css, option, select, title, wurfl_device_id, xmlpidtd.

Analysis WALL markup termination conventions:
An important difference between (c)HTML and XHTML-MP/WML-1.x is that XHTML-MP and WML-1.x adhere to xml standards and that it is required that all tags are properly terminated; improper termination can lead to faulty rendering. WALL adheres to the xml standards and requires proper termination of tags as i.e. <tag-element>…</tag-element>. The
 tag nicely illustrates the small differences between the markups that are annoying. See section 4.8 for an illustrative example and the WALL solution for this problem.

[12] The wml-1.x markup can represent some problems

Analysis general WALL markup constructs:
The example below shows the required layout of every separate WALL page i.e. every separate dialogue in TPCC.

```
<%@ taglib uri="/WEB-INF/tld/wall.tld" prefix="wall" %><wall:document>
<wall:xmlpidtd />
<wall:head>
  <wall:title>Some Title</wall:title>
 </wall:head>
<wall:body>
.....
.....
<wall:body/>
<wall:document/>
```

Example.jsp

- <%@ taglib uri="/WEB-INF/tld/wall.tld" prefix="wall" %><wall:document>

The start of a jsp page has to include the wall tag library to be able to use wall in the page.
The <wall:document> construct replaces the <html> tag and indicates the start of a page; this tag needs (convention within WML) to be on the same line on the right for support of WML-1.x.

- <wall:xmlpidtd />

This wall construct (Mnemonic: XML Processing Instruction and DTD) makes sure that the right XML headers are produced for the markup that is being generated. Just remember to put it there and be happy [15].

- <wall:head>
- <wall:title>*Some Title* </wall:title>
- </wall:head>

This general wall termination convention in combination with the <wall:xmlpidtd> tag ensures that the document title is depicted correctly on every device[13].

- <wall:body>...<wall:body/>

This general wall tag encapsulates the content of the page and has an equivalent in html. This construct should be placed in the PoC as well. Every separate page in WML and XHTML-MP has to be enclosed by the <body>...</body> tag and is also a convention in WALL.

The WALL constructs indicated with a bullet are all general constructs which should be located in every separate page i.e. can be modeled in TPCC for every separate dialogue.

Analysis scriptlets and JSTL:
Jsp's in combination with WALL is (can) be incorporated with the Java Standard Tag Library (JSTL) and has no side-effect on scriptlets <% ... %> and TPCC.

Analysis WML compatibility model:
WALL supports WML-1.x, XHTML-MP, and cHTML. Support for WML-1.x can be tricky and is done through the so called WML compatibility mode. This WML compatibility mode requires that complex WALL tags like <wall:form> are extended with the <...enable_wml="true"> property. In this analysis we will enable WML compatibility mode to try to support as many devices as possible [NFR02]. The WML compatibility mode also requires that every tag in for example a <form> becomes a WALL tag, including tags that are part of the intersection.

Note: WALL will only produce a WML deck with one[14] single card.

---

[13] In the PoC no head tag was used but this probably should have been done.

Analysis WURFL device capabilities usage with the JSTL:
The WURFL database is a 500x100 matrix, existing out of about 500 mobile devices with about 100 different capabilities. The predefined wall tags automatically incorporates this matrix to test the specific device capabilities, for example which markup should be used in a <wall:form> tag.
The <wall:load_capabilities> tag lets you use the JSTL to build expressive conditional Jsp's, a predefined *test* variable can be used for simple on/off tests on available device capabilities. The WURFL returns a Boolean, Integer or a String. In section 7.6 the usage of the WURFL database is illustrated with examples.

The <wall:load_capabilities> tag itself is a general construct which can be incorporated in a jsp page if additional tests are necessary.

7.2 Login source code

The source code below is the WALL implementation of parts of the login use-case from section 6.14. The analysis will focus on how and if this source code will be rendered correctly for different devices and if functionality like a button push will indeed submit the login information. A graphical illustration which is related to the login <form> can be seen in Figure 31, Figure 32 and Figure 33, it is assumed one can see the relevance here.

```
<%@ taglib uri="/WEB-INF/tld/wall.tld" prefix="wall" %><wall:document>
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="page"/>
<wall:xmlpidtd />
<wall:head>
  <wall:title> ecrmapplicationgeneraldialogue </wall:title>
 </wall:head>
<wall:body>
<jsp:include page="loginCreateMethodRequest.jsp"/>
<jsp:include page="logoutDoMethodRequest.jsp"/>
<wall:body/>
<wall:document/>
```

eCRMapplicationGeneralDialogue.jsp

```
<% if (so.loginPreCondition()){%>
<wall:form action="
loginPrepareMethodRequestDialogue.jsp",
enable_wml="true">
   <wall:p>
   <wall:input type="submit" value=Login/>
   <wall:p/>
</wall:form>
<% } %>
```

loginCreateMethodRequest.jsp

```
<% if (so.logoutPreCondition()){%>
<wall:form action=" logout.jsp" enable_wml="true">
   <wall:p>
   <wall:input type="submit" value=Logout/>
   <wall:p/>
</wall:form>
<% } %>
```

logoutDoMethodRequest.jsp

Analysis general constructs and conventions:
The set of HTML tags like <head><title><body><form> used in the PoC all have a WALL tag equivalent. The <html> tag can be replaced by the <document> tag and the tag from the PoC needs to be properly closed by .

[14] WML 1.X uses the deck with multiple cards for splitting up pages. This construct has been dropped in WALL and WML2.0.

Analysis general (submit) button constructs:
An important aspect with respect to handling forms is the submit button <input type="submit"> which is honored by all XHTML-MP and cHTML devices but is not supported on most old WML-1.x devices. The WALL solution just requires that the form tag is extended with the WML compatibility mode <... enable_wml="true"> and the <wall:input type> tag. WALL will automatically insert soft-key support for all WML-1.X devices. Within wml compatibility mode it is required that all UI element tags are extended with the <wall:tag> construction, the <p> tag will become <wall:p> although it is suspected <p> to be part of the intersection and would be honored by every device.

This is a general construct that can be used for every UI element and should present no problems for TPCC. It is not true that every UI tag element has to be extended with the wall tag i.e. the intersection of different markup languages is not empty. [15] Advises to drop support for WML-1.x. A form with a submit button would become see code below.

```
<wall:form action=" loginPrepareMethodRequestDialogue.jsp">
   <p>
   <input type="submit" value=Login/>
   <p/>
</wall:form>
```

Note: WML browsers from Openwave have great support for soft keys (which makes entering data quick and intuitive), while trying to program soft keys will make users' lives harder on other WML browsers. WALL fixes this transparently by programming soft keys for UP.browser 4 and providing a submit anchor to other devices. This is achieved by looking at the WURFL `softkey_support` *capability for WML devices* [15].

7.3 Long list of devices source code

The implementations of the use-cases from the PoC only use very basic constructs that have a WALL equivalent or are part of the intersection. It is expected that a long list of devices due to differences in screen size, could present some problems on some mobiles. Mobiles with large screens for example could allow a list of 6 elements to be presented where a small mobile only has room for 5 elements.

productListsElements.jsp	productObjectPreviewer.jsp

Productlist: </wall:br>
 	
<%
for (int i=0; i<co.getVectorLength(); i++) {
%>
<jsp:include flush="true"
page="productObjectPreviewer.jsp">
<jsp:param name="id" value="<%= i %>"/>
</jsp:include>
<%
}
%>
 	
	<%@ page import="java.util.*" %>
	<%@ taglib uri="/WEB-INF/tld/wall.tld" prefix="wall" %>
	<jsp:useBean id="co" class="ecrm.CatalogueObject" scope="page"/>
	<jsp:useBean id="po" class="ecrm.Product" scope="page"/>
	<%
	String idstring = request.getParameter("id");
	int id = Integer.parseInt(idstring);
	po = co.getProduct(id);
	%>
	   <wall:a href="productPresentObjectDialogue.jsp?id=<%= id %>">
	<%= po.getDescription()%></wall:a >

Analysis general constructs and conventions:
The <ol> and [15] tags are suspected to be part of the intersection and do not require any specific wall tags. The anchor tag <a href> is supported by every device as well but is not part of the intersection due to the fact

[15] and <li> are part of xhtml-mp and wml-1.x but might not be part of chtml.

that it has a different meaning inside/outside a menu element. The precise consequences of this different meaning are unclear but the usage of the <wall:a href> tag should solve any problem.

Note: This code is also expected to work on WML-1.x devices, the constructs are part of the intersection and <a href> has a wall equivalent that doesn't require the specific <...enable_wml="true"> property.

Analysis long list:
A long list is handled by wall through automatically inserting a scrollbar (if required by some screen) in the right part of the screen (see Figure 29 for a graphical illustration). Scrollbars are available on every mobile device i.e. part of the intersection. It is suspected that either the <body> or <ol> tag is able to insert scrollbars. Another option for showing a list of elements (suggested by the WALL tutorial) is the usage of the <menu> tag; this is illustrated in the next section.

7.4 Device flexibility (menu, stylesheets and picture icons)

Instead of using an ordered list <ol> tag, one could also use for example the <wall:menu> tag construct for showing a long list of hyperlinks. Every mobile supports some sort of basic menu structure but the implementations all differ a little. The <wall:menu> tag construct overcomes these differences by querying the WURFL database and implementing different preferred solutions for different devices. The <wall:menu> tag also allows for usage of the WURFL capability database with respect to the availability of using stylesheets and pictures, i.e. it tests if stylesheet/picture-format support is available on the device. The example source code below has been taken from [15] and is illustrated with pictures in (see Figure 26, Figure 27, Figure 28) from section 7.4.1.

7.4.1 Menu + stylesheet source code

```
<wall:head>
  <wall:title>...</wall:title>
  <wall:menu_css />
</wall:head>
<wall:body>
 <wall:menu colorize="true" autonumber="true">
  <wall:a href="http://url1" title="Games">Games</wall:a>
  <wall:a href="http://url2" title="Horos">Horoscopes</wall:a>
  <wall:a href="http://url1" title="Kids">Kids</wall:a>
  <wall:a href="http://url2" title="Movies"><wall:b>Movies</wall:b></wall:a>
  <wall:a href="http://url1" title="Music">Music</wall:a>
  <wall:a href="http://url2" title="Radio">Radio</wall:a>
  <wall:a href="http://url2" title="TV">TV</wall:a>
 </wall:menu>
</wall:body>
```

- <wall:menu_css />

If a mobile device has style sheet support WALL will make sure it gets delivered to the appropriate device.

This is a general construct which can be added to every separate page without affecting the TPCC approach. If a device has no css support it will be ignored, therefore it can be made part of the template.

- <wall:menu colorize="true" autonumber="true">

Within the body tag the <wall:menu>...</wall:menu> tags can replace the <ol>... tags for productListsElements and be given color for a nice look. WALL actually implements the ... tag construct for simple XHTML-MP devices and implements a <table><tr><td>...</td></tr></table> with stylesheet information through the <...colorize="true"> property for advanced XHTML-MP devices [15], for WML-1.x devices it implements a <select><option> tag.

The <..autonumber="true"> property of the menu tag inserts a number in front of the list of items and it also enables fast access keys for XHTML-MP devices, an element can be selected by pushing the right number on the keypad.

This <wall:menu> tag could be a general construct if the TPCC metamodel would incorporate Menu Elements instead of List Elements.

Figure 26

Menu on simple WML devices and primitive XHTML MP devices.

Figure 27

Menu on XHTML devices with decent table and CSS support.

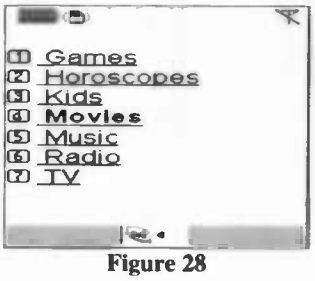

Figure 28

Menu on Imode CHTML devices.

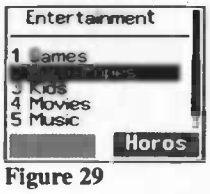

Figure 29

Menu on devices featuring UP.Browser 4 by Openwave (WML 1.X).

7.4.2 Inserting pictures in a menu

WALL <menu> also allows for the <menu_elements> tag which can be used for inserting icons, the picture is simply rendered out for devices that do not offer picture support [29]. The example below has been taken from [29] which preceded the use of WURFL, the used <menu_elements> which allows for easy insertion of pictures is no longer available in WALL. It is expected that the same functionality is implemented by the <wall:a href> tag in WALL.
The usage of menu_elements is illustrated in the source code below and graphically illustrated in Figure 30 and Figure 31.

```
<wall:body>
  <wall:menu colorize="true" autonumber="true">
    <wall:menu_item title="Weather" href="#weath" text="Weather" icon="sun"/>
```

```
.....menu_items.....
<wall:menu colorize="true" autonumber="true">
</wall:body>
```

Figure 30

Figure 31

7.5 Device flexibility (non JSTL)

The PoC from chapter 7 is so basic that WURFL can handle it with some minor chances. A problem can be created by using a <form> which uses radio-buttons. Radio buttons are not supported by every mobile and posting data is done in different formats between WML-1.x and XHTML-MP/cHTML devices.
The wall solution replaces the radio-button solution with a <select> <option> possibility which is available on every device and is able to post the entered data in an fixed format.

```
<%@ taglib uri="/WEB-INF/tld/wall.tld" prefix="wall" %><wall:document>
<wall:xmlpidtd />
<wall:head>
  <wall:title>Form 2</wall:title>
</wall:head>
<wall:body>
 <wall:form action="url" method="post" enable_wml="true">
Pin Code:
  <wall:input type="text" name="pincode" value="" format="NNNN" maxlength="4" />
<wall:br />
Choose:
  <wall:select title="Day" name="day">
    <wall:option value="11/28/04">Yesterday
    </wall:option>
    <wall:option value="11/29/04" selected="selected">Today
    </wall:option>
    <wall:option value="11/30/04">Tomorrow
    </wall:option>
  </wall:select>
<wall:br />
 <wall:input type="hidden" name="session" value="gfsa87837" />
 <wall:input type="submit" value="Go" />
 </wall:form>
</wall:body>
</wall:document>
```

Form2.jsp

Analysis general constructs:

- A <form> with radio-buttons is not supported by every device i.e. WML-1.x devices. WALL replaces the radio button with a <select><option>...</option></select> tag which can express the same functionality and uses the same data format for posting data between the mobile and the receiving service (server).
The <select><option> construct is a general construct that abstracts away from the device flexibility, but it does limit the available options for representing the UI.

Alternatives:
One alternative could be dropping support for WML-1.x devices (as suggested by the WALL tutorial) which harms [NFR02].

Another alternative that could be used is the usage of the WURFL database. This option still requires the <select><option> construct but it does allow for the representation of a <select> tag as a radio button. The WURFL database has the **capabilities.xhtml_select_as_radiobutton** method which returns a Boolean. This method can test if radio buttons are available for a device and present the <select><option> tag as a radio button.
For TPCC this could mean that the <select> <option> is the general construct and that the radiobutton is a specialization of this general construct. This means that at design time two alternatives are generated comparable to the login-precondition from the PoC and protected by a capabilities test.

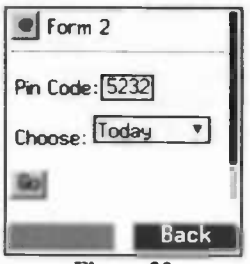

Figure 32

Form 2 on XHTML MP Device.

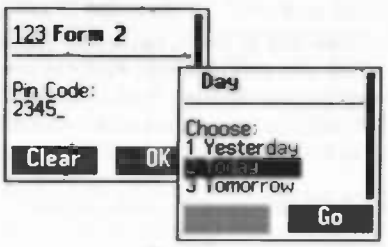

Figure 33

Form 2 on WML Browsers from Openwave (UP.Browser 4 family).

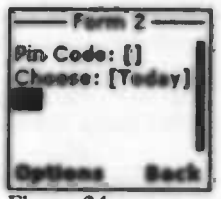

Figure 34

Form 2 on non-Openwave browsers.

7.6 Device flexibility (JSTL)

WALL also offers support for device flexibility with respect to capabilities other than layout and screen size. For example some mobiles support some version of j2me. J2me can be used for advanced services like games. Wall can easily be mixed with the Java Server Pages Standard Tag Library (JSTL). The JSTL (see section 4.10) allows for the encapsulation of core functionality like iterations or conditional tags. The mixing of JSTL and WALL is best illustrated with an example.
In the example below, the page will be rendered with a suitable menu markup for the requesting device. In addition to that, only the links to services that make sense for that device are rendered. In other words, owners of non j2me-capable devices are not presented with links to j2me services.

```
<%@ taglib uri="/WEB-INF/tld/wall.tld" prefix="wall" %><wall:document>
<wall:xmlpidtd />
<%@ taglib uri="/WEB-INF/tld/c.tld" prefix="c" %>
<wall:load_capabilities />
<wall:head>
  <wall:title>Entertainment</wall:title>
  <wall:menu_css />
</wall:head>
<wall:body>
 <wall:block>
<c:choose>
 <c:when test="${capabilities.gif}">
  <img src="pingular.gif" alt="Pingular" />
 </c:when>
 <c:otherwise>
  <img src="pingular.wbmp" alt="Pingular" />
 </c:otherwise>
</c:choose>
 </wall:block>
 <wall:menu colorize="true" autonumber="true">
<c:if test="${capabilities.midp_10}">
  <wall:a href="http://url1" title="Games">Java Games</wall:a>
</c:if>
<c:if test="${capabilities.midp_10} && ${capabilities.j2me_colors > 8}">
  <wall:a href="http://url1" title="Games">Cool Java Games</wall:a>
</c:if>
<c:if test="${capabilities.wap_push_support}">
  <wall:a href="http://url2" title="Sport">Real-Time Sport Updates</wall:a>
</c:if>
<c:if test="${capabilities.receiver}">
  <wall:a href="http://url2" title="MMS">Cool Pics</wall:a>
</c:if>
<c:if test="${capabilities.sender}">
  <wall:a href="http://url2" title="MMS">Send MMS to your friends</wall:a>
</c:if>
  <wall:a href="http://url2" title="News">News</wall:a>
 </wall:menu>
</wall:body>
</wall:document>
```

Capabilities

- <wall:block>...</wall:block>

This general construct can be used to reserve some part of the screen for including pictures depending on device capabilities.

- <c:if test="${capabilities.midp_10}">
- <wall:a href="http://url1" title="Games">Java Games</wall:a>
- </c:if>

This general construct allows for the testing of device specific capabilities: midp_10 is a version of j2me.

The position of these general constructs with respect to TPCC is unclear but it seems that things like this could be generated without to much problems. It appears that the TPCC can be extended with these constructs through OCL statements. If for example some specific eCRM functionality would require J2me capabilities the test construct can easily be generated out of OCL.

Table 5 Device capability illustration

Figure 35

Figure 36

Figure 37

Figure 38

Nokia 7110 (WML 1.1, GSM EU)

Nokia 7210 (WML 1.2, GSM EU/US)

Siemens SL45 (WML 1.1 GSM EU)

Siemens SL45i (WML 1.1 GSM EU, software upgrade of SL45 which enables J2ME)

Figure 39

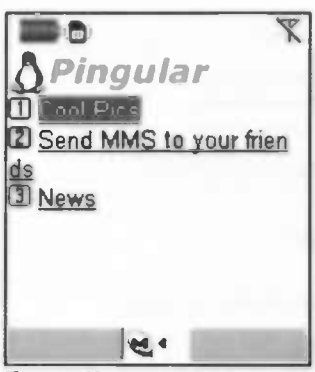

Figure 40

Figure 41

Figure 42

7.7 Evaluation WURFL/WALL

From a practical point of view WURFL/WALL seems like a simple and practical solution for handling the device flexibility. WALL significantly simplifies making an UI for multiple devices [NFR02] i.e. WALL allows for one presentation if one adheres to the mentioned WALL conventions and constructs. Differences like xml termination, submit/softkey support and radio button support are handled elegantly by using WALL and significantly simplifies the variability issues related to the presentation (device flexibility [NFR02]).

The usage of the WURFL database in combination with JSTL allows for simple on/off tests for certain device capabilities like j2me or wap-push.
The solutions and constructs which are used for constructing a one View interface by WALL cleanly abstract away from [NFR02] and appear to have no complex solutions which cannot be included by TPCC.
The tests on the WURFL database can be done through simple *if then else* constructs and also abstract away from the device specific variability. It appears to be simple to incorporate these tests into TPCC by using for example OCL constructs.

The PoC is too simple to test any weaknesses and further research is required. Below some questions are listed that have to be resolved in a further analysis.

- The support of WML-1.x is supposed to be tricky; can complex nested UI elements still be generated? I.e. can there be found a general construct?
- What is the performance of WALL?
- Can the <select><option> construct in combination with the **capabilities.xhtml_select_as_radiobutton** be used to exploit device capabilities as suggested in section 7.5.

8 Future work

Due to the large size of the total problem domain many parts of the research have been placed in this chapter. In this section some problems and possible solutions are suggested for further research.

8.1 Generators

The most important thing that has to be done in the near future is the building of the required generators for the presentational part of the PoC (PoC part one see Figure 21). There are two kinds of generators that need to be built; first the extended PIM has to be transformed into the Control and Presentation model through a model transformation language (MTL), secondly the required jsp code has to be generated from the Control and Presentation model.

Extended PIM → Control and Presentation model:
What we have is the extended PIM, the graphical representation and the TPCC meta-model. The graphical representation parameterizes the Control and Presentation model; the extended PIM has knowledge about the kind of method that needs to be executed. The expectation is that the combination of these entities can supply the required information for constructing the Control and Presentation model.

The idea is that the kind of method in the extended PIM can be linked to some control meta-types which can be used as a starting point. For example a method with parameters can be linked to CreateMethodRequest and PrepareMethodRequestDialogue but not to DoMethodRequest. The starting points can be expanded by using other properties (like the number of parameters) of the kind of method but are restricted by the properties of the TPCC meta-model. This way the complete set of possible Control and Presentation models can be created from the properties of the method in the extended PIM. The set of possibilities can be reduced by the graphical representation i.e. the graphical representation parameterizes the C & P models and reduces the possibilities. Note that it is very well possible that there is more then one option available.

Control and Presentation model → jsp:
For generating the jsp code from the Control and Presentation model we propose the usage of the Eclipse Modeling Framework (EMF). EMF is a modeling framework and code generation facility for building tools and other applications based on models and meta-models. The TPCC approach in this thesis assumes the availability of a template for the Tailorable Presentation Control Classes in which the variable parts can be filled in through the Control and Presentation model; this approach appears to be similar to the approach taken in EMF.

8.2 Adding the Interface logic to the Proof of Concept

In the Proof of Concept the elaborated model has been split in three parts; the third part for completing the PoC (see section 6) is the addition of an interface logic component between the business logic and presentation layer. The implementation of this interface logic component is going to be done through Web Services in the form of a Web Service Description Language (WSDL) interface.
The proposed solution in this thesis for incorporating the WSDL interface is the automatic generation of one WSDL interface per Java-class (see section 4.12) by using the glassfish wsimport (wsdl generation) tool (see section 4.11). This indirect wsdl generation approach does not require a separate WSDL PSM for creating the interface logic but can be indirectly generated from the extended PIM.

In section 5.2 it was mentioned that there might be a mismatch between stateful objects and stateless web services. The proposed solution for this is modeling additional methods in the extended PIM that supply this stateful information; this approach is similar to modeling the control methods in the TPCC approach.

8.3 Thin/Thick client approach

In the design decision chapter (see sections 4.5) we have chosen a thin client approach. The main reason for this was that both device flexibility [NFR02] and functional flexibility [NFR03] had a higher priority than requirements performance [NFR09] and usability [NFR10] otherwise a thick or rich client approach would have been chosen. This section however does indicate that the subject of thin/thick and intermediate clients is very broad and needs further research. One of the main concerns expressed in section 4.5.1 is the performance of the thin client approach on mobile networks. Mobile networks suffer from high latency and if the eCRM functionality requires a lot of back and forth communication (which can be expected) between client and server this might lead to an unacceptable performance of the application.

The main question that needs to be answered is: "Is a thin client approach suitable for handling the large domain of the eCRM functionality with respect to the performance?" The answer to this question is complex, the list below shows some hints on sub questions that might be valuable.

1. What is the performance of a thin client on mobile networks (with high latency properties)?
2. Does the domain of the eCRM functionality require a lot of back and forth communication between client and server?
3. Do solutions for the Unreliable connection [NFR01] require a lot of back and forth communication?
4. Do long running transaction [NFR07] require a lot of back and forth communication?

If the answer to the main question is negative then an alternative has to be found. In section 4.5 it is mentioned that an ultra thick client is not a feasible solution; however there are intermediate solutions that can improve the performance of the application. Below two suggestions are made for intermediate solutions.

1. The first suggestion is a rich client approach similar to Asynchronous Javascript and XML (AJAX); this solution uses asynchronous messaging that improves the performance of an application in unreliable (mobile) networks. Most mobiles have an equivalent to Javascript called WML-script or ECMA-script which can be used for asynchronous messaging in a similar way.
2. The second suggestion is a manageable thick client approach; this approach uses a property from MDA that allows for a manageable approach towards the addition of different mobile platforms.

8.4 WURFL/WALL and multiple views

The analysis of WALL in combination with the developed PoC did not show the need for creating multiple views for different devices; however it seems possible that in a later stage this is required. The usage of the dialogue flow notation (DFN) [48] could solve this problem. This approach uses a thin client approach that can be used for developing multiple views of the same application logic. The DFN uses different sets of possible dialogues for handling multiple views.

Can the dialogue flow notation be used in combination with WURFL/WALL and the TPCC approach?

It appears that the splitting up of a screen as is done in the dialogue flow notation is similar to a control meta-type. However there is need for a run-time detection mechanism in the control meta-type for being able to discriminate between different options. It appears that at design time the control meta-types can be extended by ocl-statements that query the WURFL data-base and make a distinct choice between available dialogue options based on this the result of this query.

8.5 Non-functional requirements

Some non functional requirements that were not handled in this thesis are: the functional type reliability [NFR05], eCRM QoS metrics [NFR06].

A further research on these non-functional requirements could give a lot of clues on a thin/thick approach.

For example further research on the subject of functional type reliability [NFR05] and eCRM QoS metrics [NFR06] give better insight in the domain of the eCRM functionality. This insight allows for the categorization of eCRM functionality which can be used for making a choice between a thin/thick and rich client approach. For example if it appears that there is lot of eCRM functionality with a high risk of hurting the relation with the customer if something goes wrong then mechanisms have to be created for minimizing this risk. If it appears that this risk can only be solved by a lot of back and forth communication between the client and server then this might indicate the need for a thick client approach.

9 Conclusion

Electronic Customer Relationship Management (eCRM) is a large enterprise application domain with complex user interaction. The dynamic market of Mobile devices introduces a lot of variability with respect to capabilities like screen-size, operating system, network support and many others. The combination of these two areas into one application makes maintainability both very important and difficult.
This thesis shows an approach that divides the total maintainability of the application into three parts; design time, static and run-time. For the design time variability it uses a model-driven development (design time) approach called Model Driven Architecture (MDA) that is used for refining a software design by generating the platform specific code in the context of a (static) 3-tiered architecture from one high-level abstraction model (PIM) through intermediate Platform specific models (PSM's). This specific usage of MDA allows for direct manipulation of the application objects through the UI elements and increases the consistency between the different layers.

The research shows that the refinement step from one high-level abstraction model (PIM) to the development of the presentation tier model (PSM) is complex. Double stereotyping, mixing of control and presentation issues in the presentation tier and the lack of application state information in the PIM hinder the usage of simple stereotypes and ocl-statements.

For solving the problems mentioned above this thesis introduces Tailorable Presentation Control Classes (TPCCs) which is a new form of model-driven User Interface development. TPCC seeks an optimum with respect to three aspects: simple modeling techniques, well-defined decoupling between UI and application objects, and enabling direct manipulation of business objects. TPCCs offer standard forms of interaction (control meta-types) between the user and the application objects which can be formalized in simple models. TPCCs strictly decouple between UI and application objects and can be model driven tailored to directly link between presentation and the application objects.
The main strength of TPCC is that it indeed finds an optimum between UI-development techniques in which a UI directly manipulate the application objects but are not model-driven such as Naked Objects [30] and technologies that do support model-driven UI development but require powerful complex models such as task modeling [40]. Technologies like naked object increase the consistency of an application through direct manipulation but cannot be used for a model-driven UI development due to the fact that required information cannot be formalized in a simple model because it is not present at a specific tier. Model-driven User Interface techniques like task modeling also use an additional layer (decoupling) between the presentation and application objects but the models in this layer are overly complex due to the fact this technique is not based on direct manipulation of application objects and therefore these models have to capture all the possible navigation and composition information, these powerful complex models undermine the basic idea behind models which should raise the level of abstraction.
One of the reasons that TPCC finds this optimum is that TPCC places part of its control in the business logic through pre-and post-conditions; this enables the application objects to steer the presentation objects which directly manipulate them, but the applications objects retain control over what may happen to them. This allows the User Interface model to remain thin and still use the information that is not available in the presentation tier. TPCC uses the PAC-pattern for decoupling the presentation from the business logic through a recursive control mechanism. The difference with PAC is that TPCC does not use a hierarchy of agents. The combination of the recursive control from PAC in combination with the direct manipulation application development from one model implies that this mapping is so direct that a hierarchy of agents might not be required.

A current shortcoming of the work on TPCC is that the required generators for developing the application still have to be built. However the source code analysis in this thesis shows a direct 1:1 relation between the developed source code and the instantiation of the TPCC meta-model. A relation between model elements is therefore directly reflected as a relation in the code, therefore it appears to be feasible that the usage of the Eclipse Modeling Framework (see section 8.1) can be used for this purpose.

Finally this thesis shows through a source code analysis that a one-view runtime implementation through WURFL can be used in combination with TPCC. WURFL is a very practical straight forward approach for supporting the variability in the field of mobile devices and implies that complex mechanisms (device variability models) as used in other model-driven User Interface development approaches [40] for porting too many different devices might not be required.

10 References

[1]. I. Jorstad, D. Dustdar, D. van Thanh, An analysis of current mobile services and enabling technologies.
[2]. http://en.wikipedia.org/wiki/Customer_relationship_management
[3]. http://en.wikipedia.org/wiki/ETOM
[4]. E. J. Wiersema, An eCRM gap analyis applied on Atos Origin TUM client groups.
[5]. http://ws.apache.org/muse/
[6]. http://ws.apache.org/wsrf/
[7]. http://ws.apache.org/axis2/
[8]. http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html
[9]. http://mobdev.tknerr.de/2006/08/27/web-services-resource-framework-wsrf-for-j2me-version-01/
[10]. http://grids.ucs.indiana.edu/ptliupages/publications/CGL-WebServices-Chapter.pdf p11-12 ws-context
[11]. http://www.gentleware.com/
[12]. Gartner, How to Be Successful With Enterprise Mobility.
[13]. http://developer.openwave.com/dvl/support/documentation/guides_and_references/xhtml-mp_style_guide/appendixb.htm
[14]. http://wurfl.sourceforge.net/
[15]. http://wurfl.sourceforge.net/java/tutorial.php
[16]. Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York, NY: 1990.
[17]. http://msdn2.microsoft.com/en-us/library/ms978678.aspx
[18]. http://msdn2.microsoft.com/en-us/library/ms978689.aspx
[19]. http://msdn2.microsoft.com/en-us/library/ms978748.aspx how-to use mvc http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html
[20]. http://www.phptr.com/articles/article.asp?p=169547&seqNum=6&rl=1
[21]. http://en.wikipedia.org/wiki/Interoperability
[22]. Model Driven Architecture (MDA) Document number ormsc/2001-07-01.
[23]. R. B'Far Mobile Computing Principles Designing and Developing Mobile Applications with UML and XML Cambridge University.
[24]. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture A system of Patterns.
[25]. Coutaz, PAC, an Object Oriented Model for Dialog Design.
[26]. J.Coutaz, PAC: An Object Oriented Model for Implementing User Interfaces, SIGCHI bulletin, Vol 19, No2, pp. 37-41, 1987.
[27]. http://en.wikipedia.org/wiki/Model-driven_architecture.
[28]. T Glover, J Davies. Integrating device independence and user profiles on the Web.
[29]. http://oui.sourceforge.net.
[30]. R. Pawson and R. Matthews, Naked objects: a technique for designing more expressive systems, ACM SIGPLAN Notices, vol 36, nr 12, pp 61-67, ISSN 0362-1340, ACM Press, New York, USA, 2001
[31]. http://en.wikipedia.org/wiki/Platform-specific_model
[32]. http://en.wikipedia.org/wiki/Platform-independent_model
[33]. http://en.wikipedia.org/wiki/Object_(computer_science)
[34]. http://en.wikipedia.org/wiki/Service-oriented_architecture
[35]. http://www.omg.org/cgi-bin/doc?formal/06-05-01
[36]. http://en.wikipedia.org/wiki/Thin_client
[37]. http://www.gogis.nl/en/tutorial/docs/J2EE/index.html
[38]. http://en.wikipedia.org/wiki/Wireless_Markup_Language.
[39]. http://en.wikipedia.org/wiki/JavaServer_Pages
[40]. P.A. Muller, P. Studer, J. Bézivin, Platform Independent Web Application Modeling, P.Stevens et al. (Eds.): UML 2003, LNCS 2863, pp. 220–233, 2003.

[41]. G. Mori, F. Paterno, C. Santoro, CTTE: Support for Developing and Analyzing Task Models for Interactive System Design, IEEE Transactions on Software Engineering, pp. 797-813, 2002

[42]. P. Fraternali and P. Paolini, Model-driven development of Web applications: the AutoWeb system, ACM Trans. Inf. Syst., Vol. 18, No. 4, pp. 323—382, 2000

[43]. S. Ceri, P. Fraternali, A. Bongio, Web Modeling Language (WebML): a modeling language for designing Web sites, Computer Networks, Vol. 33, No. 16, pp. 137—157, 2000

[44]. J. Vanderdonckt, Q. Limbourg, B. Michotte, L. Bouillon,D. Trevisan, M. Florins, USIXML: a User interface Description Language for Specifying Multimodal User Interfaces, in Proc. of W3C Workshop on multimodal Interaction WMI'2004

[45]. D. F. Gieskens, J. D. Foley, Controlling user interface objects through pre- and postconditions, CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 189—194, 1992

[46]. R. Pawson and R. Matthews, Naked objects: a technique for designing more expressive systems, ACM SIGPLAN Notices, vol 36, nr 12, pp 61-67, ISSN 0362-1340, ACM Press, New York, USA, 2001

[47]. A.Weinand, E. Gamma, R. Marty, ET++ An Object-Oriented Application Framework in C++, OOPSLA 88 proceedings, pp. 46-57, 1988.

[48]. Matthias Book and Volker Gruhn, A Notation and Framework for Dialog Flow Control in Web Applications

Appendix A

**Code snippet Wurfl/WALL
 tag hangling**

```
//get the user agent
UA = TagUtil.getUA(this.request);

device_id = uam.getDeviceIDFromUALoose(UA);

markup = cm.getCapabilityForDevice(device_id,
                                   "preferred_markup");

//XHTML <br/> with trailing slash
 if ( markup.indexOf("xhtmlmp") != -1) {
   try {
       JspWriter out = pageContext.getOut();
       out.println("<br/>");
     } catch(IOException ioe) {
       System.out.println("Error in br tag: " + ioe);
     }
     return(SKIP_BODY);
 }

 //CHTML <br/> without trailing slash
 if ( markup.indexOf("chtml") != -1) {
   try {
       JspWriter out = pageContext.getOut();
       out.println("<br>");
     } catch(IOException ioe) {
       System.out.println("Error in br tag: " + ioe);
     }
     return(SKIP_BODY);
 }
```

Code snippet example from wurfl.xml

```
<device user_agent="NokiaE60" actual_device_root="true"
fall_back="nokia_generic_series60_dp30" id="nokia_e60_ver1">
        <group id="product_info">
                <capability name="model_name" value="E60"/>
        </group>
        <group id="display">
                <capability name="resolution_width" value="352"/>
                <capability name="resolution_height" value="416"/>
                <capability name="max_image_width" value="352"/>
                <!-- Antonio Martinez: supposed values -->
                <capability name="max_image_height" value="380"/>
                <!-- Antonio Martinez: supposed values -->
        </group>
        <group id="image_format">
                <capability name="colors" value="16777216"/>
        </group>
        <!—etc ---------- >
```

<!—etc ---------- >
</device>

Appendix B

This section describes some of the removed design decisions from chapter 4.

WS-ResourceFramework:

The WS-RF is a collection of five different specifications WS-Resource, WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup and WS-BaseFaults. They all relate to the management of WS-Resources. The WS-Resource solution for giving Web Services the ability to keep state information while keeping them stateless is done by keeping the Web Service and the state information completely separate. Instead of putting the state *in* the Web Service (i.e. making it Stateful[16]) we will keep it in a separate entity called a **resource** which will store all the state information.

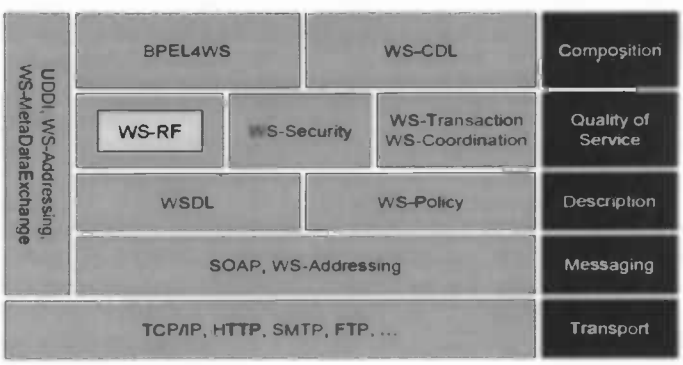

Figure 43
WS-ResourceProperties
A resource is composed of zero or more resource properties. For example, in the figure shown above each resource has three resource properties: Filename, Size, and Descriptors. WS-ResourceProperties specifies how resource properties are defined and accessed. As we'll see later on when we start programming, the resource properties are defined in the Web service's WSDL interface description.

WS-ResourceLifetime

---

16 The following link gives a good/simple example of the stateles/statefull Web Service problem http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s03.html

Resources have non-trivial lifecycles. In other words, they're not a static entity that is created when our server starts and destroyed when our server stops. Resources can be created and destroyed at any time. The WS-ResourceLifetime supplies some basic mechanisms to manage the lifecycle of our resources.

WS-ServiceGroup
We will often be interested in managing groups of Web Services or groups of WS-Resources, and performing operations such as 'add new service to group', 'remove this service from group', and (more importantly) 'find a service in the group that meets condition FOOBAR'. The WS-ServiceGroup specifies how exactly we should go about grouping services or WS-Resources together. Although the functionality provided by this specification is very basic, it is nonetheless the base of more powerful discovery services (such as GT4's IndexService) which allow us to group different services together and access them through a single point of entry (the service group).

WS-BaseFaults
Finally, this specification aims to provide a standard way of reporting faults when something goes wrong during a WS-Service invocation.

Related specifications:

WS-Notification
WS-Notification is another collection of specifications that, although not a part of WSRF, is closely related to it. This specification allows a Web service to be configured as a notification producer, and certain clients to be notification consumers (or subscribers). This means that if a change occurs in the Web service (or, more specifically, in one of the WS-Resources), that change is notified to all the subscribers (not all changes are notified, only the ones the Web services programmer wants to).

WS-Addressing
As mentioned before, the WS-Addressing specification provides us a mechanism to address Web services which is much more versatile than plain URIs. In particular, we can use WS-Addressing to address a Web service + resource pair (a WS-Resource).

Properties:
- A WS-Resource is a Stateful web service.
- The resource can be stored in a file/memory or even a database.
- A WS-Resource is the pairing of a Web Service with a resource (**Web Service + Resource = WS-Resource**).
- A WS-Resource is a distributed object
- The address of a WS-Resource called an *endpoint reference*, which is a versatile way of addressing Web Services (compared to URIs) through the use of WS-Addressing.
- A Resource is composed of zero or more resource properties.
- A WS-Resource-property is a piece of information defined as part of the state model, reflecting a part of the WS-Resource's state. WS-ResourceProperties specifies how resource properties are defined and accessed.
- The resource properties are defined in the Web Services WSDL interface description.
- WS-RF supports dynamic insertions and deletion of the Resource Properties of a WS-Resource at run time
- WS-ResourceLifetime is used for managing the lifecycle of the WS-Resource
- WS-ServiceGroup is used for managing groups of Web Services or groups of WS-Resources.
- WS-BaseFaults is a standardized way of reporting errors.
- WS-RF is based on the WSDL/SOAP/WS-Addressing/WS-Notification properties.
- WS-RF is used a lot in grid computing.

Design decision:

At this stage we are not going to use WS-RF to model Stateful Web Services. The solution of WS-RF can be used in the future if this is necessary.

Rationale:

- The choice of using eCRM functionality in the back-end in combination with the wsdl interfaces leads to a mismatch [P01] between Stateful objects and stateless web services. We expect that the use of WS-RF can solve this mismatch but this solution might be a little to complex at this stage.

Alternatives:

- WS-Context
- Direct modeling of a Stateful resource, this will probably use WS-Addressing.
- Modeling methods which return state full information.

WS-Context

WS-Context is like WS-RF a means to model state between transactional Web Services. This approach uses a Web Service to model state and doesn't use a separate resource that is decoupled from the Web Service itself.

Properties

- Separate context-service
- Conforms to the Web Service architecture i.e. not a separate resource but a separate service with clear functionality.

Design decision:
We are not going to use WS-Context.

Rationale:
Although WS-Context might be a better solution than WS-RF i.e. it is in line with the Web Service architecture. The eCRM context can get very complex, WS-Context is a complex solution itself the combination is too complex at this moment.

Dialog Flow Notation

Properties:

- Client independent representation of the UI (thin client approach).
- Good integration with MVC or Observer Pattern
- Communication through http/SOAP
- Servlets handling the selection of the right channel i.e. the different mobiles.
- HTML/XHTML templates/forms with JSP
- The use of DFN allows for thick client applications with a thin client approach.

Design decision:
At this moment we are not going to use DFN.

Rationale:

- An application that should be available on any device can virtually only be fulfilled by a thin client approach.
- We have permission of EDOC to use the Dialog Flow model they developed to separate the presentation logic from the business logic.
- MDA can be used to develop the required JSP pages (masks/forms) in the Dialog Flow Model.
- The evolution of the eCRM functionality requires that the coupling between the client and the back-end should be very low.

Appendix C

Complete source code jsp-files:

eCRMapplicationGeneralDialogue.jsp

```
<html>
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="page"/>
<title>ecrmapplicationgeneraldialogue</title>
<jsp:include page="loginCreateMethodRequest.jsp"/>
<jsp:include page="logoutDoMethodRequest.jsp"/>
<jsp:include page="registerCreateMethodRequest.jsp"/>
<jsp:include page="productsListElements.jsp"/>
</html>
```

loginCreateMethodRequest.jsp

```
<html>
<jsp:useBean id="so" class="ecrm.SecurityObject"
scope="session"/>
<% if (so.loginPreCondition()){%>
<form action="loginPrepareMethodRequestDialogue.jsp">
   <p>
     <input type=submit value=Login>
   </p>
</form>
<% } %>
</html>
```

logoutDoMethodRequest.jsp

```
<html>
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="session"/>
<% if (so.logoutPreCondition()){%>
<form action="logout.jsp">
   <p>
     <input type=submit value=Logout>
   </p>
</form>
<% } %>
   </html>
```

Logout.jsp

```
<jsp:useBean id="so" class="ecrm.SecurityObject"
scope="session"/>
<jsp:setProperty name="so" property="*"/>
<% so.doLogout(); %>
<jsp:forward page="eCRMapplicationGeneralDialogue.jsp"/>
```

loginPrepareMethodRequestDialogue.jsp

```
<html>
<%@ page import="ecrm.*" %>
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="session"/>
<jsp:useBean id="customer" class="ecrm.Customer" scope="page"/>
<jsp:setProperty name="customer" property="*"/>

<title>loginpreparemethodrequest.jsp</title>

<form action="loginPrepareMethodRequestDialogue.jsp" method="get">
   <p>
   Name:    <input type="text" name=name>
   </p><p>
   Password:<input type="text" name=password>
   </p><p>
   <input type=submit value=Submit>
   </p>
</form>

<%
      if (customer.getName()!= null){%>
<%
            if ((boolean)so.loginParameterCondition(customer.getName(), cust
omer.getPassword())){
                  so.doLogin(customer.getName(),customer.getPassword());%>
                  <jsp:forward page="eCRMapplicationGeneralDialogue.jsp"/>
<%
            }else{%>
                  <jsp:include page="error.jsp"/>
<%          }
      }
%>
</html>
```

registerCreateMethodRequest.jsp

```
<jsp:useBean id="so" class="ecrm.SecurityObject" scope="session"/>
<% if (so.registerPreCondition()){%>
<form action="registerPrepareMethodRequestDialogue.jsp">
   <p>
     <input type=submit value=Register>
   </p>
</form>
<% } %>
```

productsListElements.jsp

```
<html>
<jsp:useBean id="co" class="ecrm.CatalogueObject" scope="page"/>

<title>productsListElements</title>
<br>Productlist:
<ol>
<%
      for (int i=0; i<co.getVectorLength(); i++) {
%>
<jsp:include flush="true" page="productObjectPreviewer.jsp">
 <jsp:param name="id" value="<%= i %>"/>
</jsp:include>
<%
      }
%>
</ol>
</html>
```

productObjectPreviewer.jsp

```
<%@ page import="java.util.*" %>


<jsp:useBean id="co" class="ecrm.CatalogueObject" scope="page"/>
<jsp:useBean id="po" class="ecrm.Product" scope="page"/>

<%
      String idstring = request.getParameter("id");
      int id = Integer.parseInt(idstring);
      po = co.getProduct(id);
%>
<title>productObjectPreviewer.jsp</title>
<li> <A HREF="productPresentObjectDialogue.jsp?id=<%= id %>"><%= po.getDescription()%></A>
```

productPresentObjectDialogue.jsp

```
<%@ page import="java.util.*" %>
<%@ page import="ecrm.*" %>
<jsp:useBean id="co" class="ecrm.CatalogueObject" scope="page"/>
<jsp:useBean id="product" class="ecrm.Product" scope="page"/>
<title>productPresentObjectDialogue.jsp</title>
<%
      String idstring = request.getParameter("id");
      int id = Integer.parseInt(idstring);
      Product p1 = new Product();
      p1 = co.getProduct(id);
      product = p1;
%>
description: <%= product.getDescription() %>
<form action="orderPrepareOrderRequestDialogue.jsp">
   <p>
     <input type=submit value=Order>
   </p>
</form>
```

Appendix D

WSDL -template

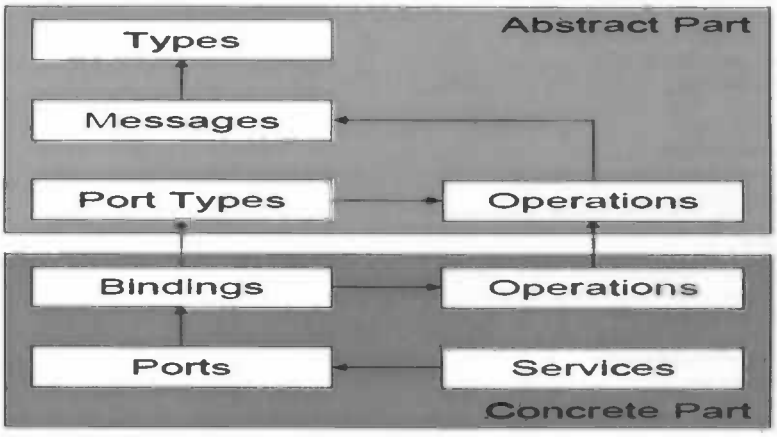

Figure 44 WSDL template

```
<?xml version="1.0" encoding="UTF-8"?>
definitions name="FactoryService"
    targetNamespace="http://www.globus.org/namespaces/examples/core/FactoryService" ❶
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.globus.org/namespaces/examples/core/FactoryService"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing" ❷
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--============================================================
                    T Y P E S
  ============================================================-->
<types>
<xsd:schema targetNamespace="http://www.globus.org/namespaces/examples/core/FactoryService"
    xmlns:tns="http://www.globus.org/namespaces/examples/core/FactoryService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        ❸
        <xsd:import
            namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
            schemaLocation="../../ws/addressing/WS-Addressing.xsd" />
        <!-- REQUESTS AND RESPONSES -->

        <xsd:element name="createResource">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="createResourceResponse">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="wsa:EndpointReference"/>
                </xsd:sequence>
        </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>
<!--============================================================
                    M E S S A G E S
  ============================================================-->
<message name="CreateResourceRequest">
```

```
        <part name="request" element="tns:createResource"/>
</message>
<message name="CreateResourceResponse">
        <part name="response" element="tns:createResourceResponse"/>
</message>
<!--============================================================
                    P O R T T Y P E
  ============================================================-->
❹
<portType name="FactoryPortType">

        <operation name="createResource">
                <input message="tns:CreateResourceRequest"/>
                <output message="tns:CreateResourceResponse"/>
        </operation>

</portType>
</definitions>
```