Design of a Sales Support System

Graduation Thesis

Alex Westerhof

February 28, 2007

RuG

Atos Origin

Design of a Sales Support System

Maximizing Customer and Supplier Satisfaction

Graduation Thesis

Author:	Alex Westerhof
Location:	Groningen
Date:	February 28, 2007
Institution:	Department of Mathematics and Computing Science, Rijksuniversiteit 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)

Abstract

The Solution and Design team of Atos Origin TUM, a company that develops CRM applications, would like to add an Internet-based sales support system for self-service to their software suite. This system should extend their existing ordering application with support for a short sales process suitable for e-commerce websites. This thesis describes the research that has been done in order to explore the possibilities of such a sales support system. The system has to support a range of devices, including PC, TV, mobile phone and PDA. The amount of personal data used has to be adjustable: the customer determines which personal data is used for the advice. The objective of the research in this thesis is to make a design and a prototype which can help Atos Origin TUM by building the system. The research and the design focus primary on the system's back-end, containing the business logic and the coupling with the front-end, which is responsible displaying the content on the user's device. The specific problem presented by Atos Origin TUM has been split up in two parts: the business aspects and the technical aspects. Both parts are studied in a context broader than just the sales support system, which has led to general solutions. The research concerning the business aspects has resulted in a short length sales process, which we call the Micro Funnel, suitable for e-commerce websites on various devices. The technical aspect of the research shows that the so called Message-based Model-View-Controller architecture is very suitable for systems that serve multiple devices over the Internet infrastructure. The solutions have partly been applied to the specific problem of the sales support system. For practical reasons part of the general solutions, have been replaced by our own design.

Preface

In order to graduate for my Computing Science study at the University of Groningen, I started with my graduation assignment at the end of 2005. I just had been spending the preceding ten weeks on an internship at Atos Origin TUM and they provided me the opportunity to graduate on a very interesting subject. I accepted this offer and my internship was prolonged for another six months.

Just like at the preceding internship I was being supervised and supported by Roelof Rabbers. I would like to thank Roelof for providing me this opportunity and for his time and support during my internship and also afterwards. Despite his busy schedule, he always found time to support me. At the university, I found Rein Smedinga and Theo Dirk Meijler willing to supervise me. Theo Dirk's input on the scientific aspects was very useful and constructive. The support of Rein primarily concerned the overall structure of this thesis. I would like to thank Rein and Theo Dirk for the time and effort they put into this assignment. I also would like to thank all the other people at both Atos Origin TUM and the University of Groningen that I have interviewed or just been asking one or more questions I had for them.

Because of the different interests of the University and Atos Origin TUM, too little attention had been paid to the scientific character of the thesis during my internship. This shortcoming was observed by Rein and Theo Dirk in the last weeks of my internship. Nevertheless, I decided to finish the design and the prototype first. After my internship, I further worked out the scientific aspects of the thesis under supervision of Rein and Theo Dirk. In spite of the fact that this cost some extra months, I found it very useful.

In front of you lies the final report that presents the results of my graduation assignment. I hope you will enjoy reading this paper.

Alex Westerhof

Groningen, February 2007

Summary

The Solution and Design team of Atos Origin TUM would like to add a sales support system for self-service to their software suite. In this way not only the ordering, but also the preceding sales process will be supported. The system has to be suitable for a range of devices, including PC, TV, mobile phone and PDA. It has to use the Internet infrastructure for communication with customers. Furthermore, it should support multiple languages and other locales. The system should be able to make a suitable, personalized offer to the customer based on various characterizations including his actual need, personal data and earlier contact. The amount of personal data used has to be adjustable: the customer determines which personal data is used for the advice.

The specific problem presented by Atos Origin TUM can be split up in business aspects and technical aspects. Both parts are studied in a context broader than just the sales support system. The research is performed by a literature review of several books and research papers. Furthermore, information from various conversations with the supervisors and other persons from Atos Origin TUM and the University of Groningen has been used.

The objective of the research is to make a design and a prototype which can be used by Atos Origin TUM to build the sales support system. The research and the design focus primary on the system's back-end, containing the business logic and the coupling with the front-end, which is responsible displaying the content on the user's device. The main research question which is answered in this thesis is as follows:

Which software architecture is suitable for the design of a self-learning system that provides personalized business-customer interaction on various devices over the Internet infrastructure considering time-to-market, development costs, flexibility, maintainability and reusability?

In our research, we are dealing with the tension between two requirements that are not often combined in current software development. The first one is

manipulation of stateful elements. In eCRM stateful elements are manipulated, i.e. customers with as state a set of products that have been selected for them, or that they have selected. To present stateful elements to users, the Naked Objects movement adopted the idea to let users directly manipulate objects. The second requirement is the need for a distributed loosely coupled architecture. Especially an architecture where the graphical representation may be decoupled from the business logic.

The Model-View-Controller (MVC) approach is a standard pattern for visualizing objects in an object-oriented system based on a direct method invocation. The model encapsulates data and rules for accessing and updating this data. The view renders the model in such a way that it can be viewed to the user as part of the user interface. Often, the controller, which processes events, and the view are combined in one user interface. There may be lots of users interacting with the model at the same time and each user interface may contain different views of the model. MVC separates model, view and controller components, increasing reusability.

To enable a distributed, decoupled architecture a service-oriented architecture (SOA) can be applied. In a SOA environment, resources in a network are made available as an interconnected set of services that are accessible through standard interfaces and messaging protocols. Services are self-describing, independent, loosely coupled pieces of functionality that perform specific functions. Using a middleware infrastructure, distributed services can be invoked and a wide range of computing devices using various software platforms can be connected. By composing services out of others, logic is divided into services, increasing reusability. Web services have become one of the most important standards for realizing SOAs and have gained broad industry acceptance. A Web service is a specific kind of service that uses the Internet for communication using open Internet-based standards.

Since traditional MVC is based on direct method invocation in an object-oriented system, it doesn't benefit much from network bandwidth improvements. Message-based MVC (M-MVC) is a SOA that uses Web services and is supposed to be a distributed version of the MVC pattern, using message-based interactions between model and view components. This enables long distance linkage between model and view.

One of the theoretical models describing the sales process is called the Sales Funnel. The original Sales Funnel idea is designed for selling complex products, like consulting services. It describes the stages of the sales process from

identifying the prospects to closing the sale. The traditional Sales Funnel is designed to be used for sales processes with personal contact between customer and seller. In our situation we are dealing with self-service. Instead of personal contact with a sales person, the customer has to use the Internet to interact with the sales support system in order to be served. Furthermore, the traditional Sales Funnel is normally used in long sales processes instead of our relatively short process. We have designed a Sales Funnel that is more suitable for self-service through the Internet infrastructure. Because the accent in our approach is on a short sales process, we call our funnel: Micro Funnel (μF).

At the top of the Micro Funnel are all the visitors of the website. At the bottom of the funnel, are the people who have bought one or more products and have completed the transaction. The Micro Funnel exists of a fixed number of steps. Depending on the demand of the customer, he either processes all the steps or only a few. The steps include identifying the customer's need, determine his main and detail choice and finally the deal registration.

Today, personalization is widely used on e-commerce websites in order to recommend products or services to customers. Personalization is the adjustment and modification of all aspects of a website that are displayed to a user in order to match that user's needs and wants. While moving through the Micro Funnel, a state indicating how much information the customer has provided about himself is kept. The content shown to the customer is based on this state, which is part of the GPI domain model. This model has three states, which lead to showing a generic, profile-based or individual selection of the portfolio.

The main component in the architecture of the sales support system is the Scenario Manager, which contains the business logic. The Content Manager forms the front-end of the system, taking care of the presentation of data to the user's devices. These two main components communicate by exchanging messages in a request/ response style.

In our menu structure concept, a menu is represented as a list. This list contains all the data that is necessary to generate a menu screen, which can be viewed by the user. The menu structure and the menus are fixed, except for the listed products. The menus are divided into three levels. On level 1 there is just one main menu, which is supposed to be a portal where the user enters the website. On level 2 are sub menus which form the product categories. On level 3 are product nodes containing information about the product in question and cross selling and up selling activities by the presence of links to other product nodes.

In order to determine which product nodes have to be added to the menus, there is a standard order in which the products are displayed for each menu. This standard order is determined by giving each menu-product combination a certain priority. If the customer has logged in or profiled before entering a particular menu, the priorities are adjusted by values corresponding to the customer. These personalized values are determined by letting the customer fill in a profiling form. In this way the customer has his own, personalized, menu structure.

Using the design, a prototype of the sales support system has been built. The prototype is based on the lay-out of the e-commerce website of one of the largest telecom operators in the Netherlands. Using the prototype both technical and business concepts were tested. The Scenario Manager is implemented as a set of Web services, which can be requested by the Content Manager using Remote Procedure Calls. The XML messages are transferred over HTTP using the SOAP protocol.

If the sales support system is going to be implemented, we recommend to look at the possibility of replacing our current messages and their communication by RPC, by a message-oriented middleware framework like NaradaBrokering. We expect that, besides a looser coupling, the messaging performance will also increase.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Atos Origin TUM (Telecom, Utilities & Media) is a company that develops CRM applications for 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 and services.

The Solution and Design team of Atos Origin TUM would like to add an Internet-based sales support system for self-service to their software suite. In this way not only the ordering, but also the sales process will be supported. This thesis describes the research that has been done in order to explore the possibilities of such a sales support system. The sales support system has to be focused primarily on companies in the telecom sector.

1.1 Initial Motive

Compared with the past, today's computers are becoming more and more non-desktop devices and this trend will continue in the future [1]. This means that different kinds of mobile devices and digital television set-top boxes will outnumber the personal computers connected to the Internet. The difference between these new devices and personal computers is that they come with varying sizes and properties. For example, different kinds of mobile devices have very different screen sizes and input mechanisms. Furthermore, according to respectively Moore's and Gilder's laws, computer processing performance and network bandwidth improves continuously. Because of these improvements, the Internet infrastructure has become very interesting for today's and future devices.

A sales support system for self-service, which utilizes these techniques, has a big advantage compared to traditional outbound marketing. It is a challenge

to do the right offer for the right customer at the right time. Because an Internet-based sales support system can be integrated in e-commerce websites that are based on inbound interactions, the company has the customer's time and attention. Having the customer's attention, the system has to do the right offer. Returning customers can be offered new products and services by using cross-selling and up-selling based on their personal data and their previous visits. When enough customers accept offers, the sales support system will contribute to both increased profits and customer satisfaction.

1.2 Requirements

Together with Atos Origin TUM, requirements for the sales support system were formulated. Atos Origin TUM would like their sales support system to be suitable for a range of devices, including PC, TV, mobile phone and PDA. The system also has to use the Internet infrastructure for communicating with customers. One of the goals of the sales support system is to let the customer and his data be the central point. Interactions between customer and company have to be known and stored in a central place. The customer should have the feeling that he is dealing with one company, even if he is communicating through different channels.

The system should be able to make a suitable, personalized offer to the customer. This offer can for example exist of products or services or can be an extension of a subscription. It has to be based on:

- Actual desire or problem of the customer.
- Personal data of the customer.
- Personal situation and characteristics of the customer.
- Earlier contact between customer and supplier.
- Complete portfolio of the supplier.

Furthermore, the system should have the following properties:

- It is IP-based.
- It supports multiple devices, including: PC, TV, mobile phone and PDA.
- The amount of personal data used is adjustable. The customer determines which personal data is used for the advice.

- The system is self-learning.
- It supports locales: properties such as language, date, time, number and currency representation can be modified.

1.3 Problem Statement

Now that the requirements of the sales support system are clear, a problem statement can be formulated. The objective of the research is to make a design and a prototype which can be used by Atos Origin TUM to build a sales support system that contributes to maximizing customer and supplier satisfaction. The specific problem presented by Atos Origin TUM can be split up in two parts: the business aspects and the technical aspects. Both parts will be studied in a context which is broader than just the sales support system. This will lead to general solutions which will be applied to the specific problem of the sales support system.

The research concerning the business aspects should include finding out which role IT can play in improving the satisfaction of both customer and company while interacting with each other. The customer shall for example be satisfied when he's served in a personal way, just like he's used to in the brick and mortar stores in his neighborhood. The selling company is satisfied when the customer is satisfied and, of course, when a lot of products and services are being sold at a profitable price.

For the technical aspect of the research investigation of which existing software architectures and design patterns can be used to realize a suitable architecture for systems that serve multiple devices over the Internet infrastructure. For example, the context, or just a part of it, of the object model containing information about customers, products, etc. has to be known by the web browser running on the customer's device. The architecture should support this. The architecture should also provide a smooth interaction (not too much delay) between clients and the server-side.

1.3.1 Research Question

The main research question which should be answered in this thesis is shown below.

Which software architecture is suitable for the design of a self-learning system that provides personalized business-customer interaction on various devices over the Internet infrastructure considering time-to-market, development costs, flexibility, maintainability and reusability?

To make this research question a little clearer, it may be useful to give some definitions. According to the *IEEE Standard Computer Dictionary* [2] the definitions of the quality attributes flexibility, maintainability and reusability are:

Flexibility The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

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.

Reusability The degree to which a software module or other work product can be used in more than one computing program or software system.

1.3.2 Research Sub-questions

To answer the research question step-by-step, it has been split up in several sub questions:

1. What architectural model is the most suitable for a system that needs to support multiple devices?
2. How can IT be used for improving the satisfaction of both parties in business-customer interaction?
3. What are the constraints when using the Internet infrastructure for communication?
4. How can the context of the business object model be kept up-to-date on the customer's web browser?
5. What are the relations between the quality attributes time-to-market, development costs, flexibility, maintainability and reusability and how do they affect each other?
6. On which specific sales process concept should the system be based?
7. What personalization mechanism is suitable for the system to use?

1.4 Research Methods

The research is performed by a literature review of several books and research papers. Furthermore, information from various conversations with the supervisors and other persons from Atos Origin TUM and the University of Groningen has been used. Details can be found in the reference list at the end of this report.

1.5 Structure of the Thesis

The rest of this thesis is structured as follows. In chapter 2 the theoretical background of the study is given. Subjects like the sales process, personalization and software architectures are introduced and an analysis is given about how these theoretical subjects can be used for the research to be performed. Chapter 3 describes the design of our sales support system, based on the theoretical foundation given in the previous chapters. First, the business aspects of the design are given. We introduce our own sales process for self-service in an IP-based environment. After the sales support system's architecture is given and there is some special attention for the communication between the two main components. The design has been implemented as a prototype, which is described in chapter 4. Finally, in chapter 5, conclusions are drawn and recommendations are given.

Chapter 2

Background

This chapter describes the background of this thesis. First it is useful to give a short introduction to the traditional sales process. In this introduction some sales terms are introduced which will be used later on and may not be familiar by all the readers. The Sales Funnel, which is a specific kind of sales process is also introduced. Furthermore an introduction is given on the personalization of e-commerce websites including a survey of techniques that are currently used. After that, some background information is given about software architectures. We will compare various existing types of software architectures and argue which is most suitable for creating a loosely coupled architecture which we are looking for.

2.1 Introduction to the Sales Process

In traditional selling, the sales process refers to a sequential series of actions by the sales person that leads towards the customer buying a product or a service. A sales process exists out of several steps, which in general sense include prospecting, contact, negotiation, fulfilling and follow-up.

A lead is a potential customer who might have a need that can be satisfied with a companies' products or services. Leads that have entered the selling phase are called prospects. If they have not yet entered the selling phase, they are called suspects.

Prospecting and follow-up can also be seen as activities apart from the actual sales process. Prospecting is part of marketing and includes locating and qualifying prospects [3].

The length of the sales cycle can depend on the cost of the product, quality of leads, ease of identifying the buyer within an organization, and the complex-

ity of the buying decisions. A normal selling cycle could be referred to as the amount of time between initial contact and the agreement to buy [31].

2.1.1 The Sales Funnel

One of the theoretical models describing the sales process is called the *Sales Funnel*. The Sales Funnel is part of the Miller Heiman approach to sales and was first described in the book *Strategic Selling* [31]. The original Sales Funnel idea is designed for selling complex products, like consulting services. It describes the stages of the sales process from identifying the prospects to closing the sale, as can be seen in figure 2.1.

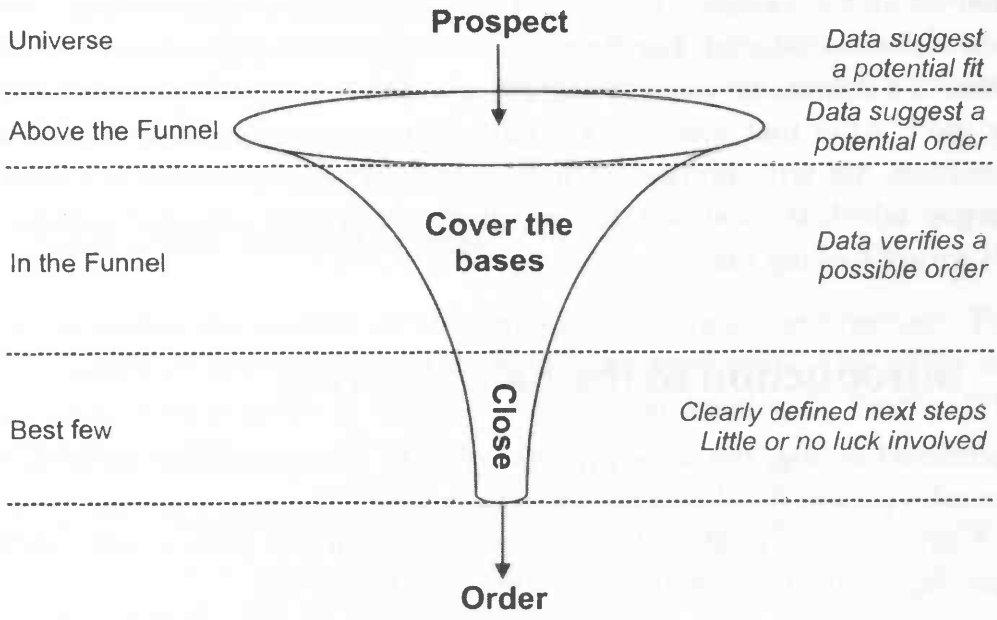

Figure 2.1: The Sales Funnel

The metaphor of a funnel is used because supply and demand are coming closer together while moving through the funnel. At the top of the funnel are a large number of prospects flowing through to a smaller number with a few closed sales coming out the bottom of the funnel. So, prospects drop out at each stage of the sales process, but the further they are in the funnel, the higher the chance is that they are going to buy something. The sales funnel can be used to manage each of the stages in the sales process. Also the overall sales process

can be managed. This avoids that the focus is only on closing some sales while there are no other sales in progress.

2.2 Personalization

In the sales process that is used on the Internet, an important aspect is *personalization*. Personalization is the selective delivery of content and services to customers and prospective customers [4]. It can also be seen a little broader and then personalization is the adjustment and modification of all aspects of a website that are displayed to a user in order to match that user's needs and wants [5]. Today, personalization is widely used on e-commerce websites in order to recommend products or services to customers [5, 6]. A system that makes such recommendations is called a *recommender system*.

There are several reasons why recommender systems are valuable for e-commerce [6]. The first reason is that people often browse e-commerce websites without buying. The recommender system can help the customer by recommending products he might like. Furthermore, the customer is now served in a personal way, just like he's used to in the brick and mortar stores in his neighborhood. When little or noting is known of a customer, non-personalized recommendations can be done. These recommendations are independent of the customer, so they are the same for each customer. They can for example be based on what other customers have said about the product or it can be the choice of the e-commerce site.

When a customer has decided to buy something the recommender system can recommend additional products. This is called cross-selling. Once the customer has decided to buy something the recommender system can also advise the customer to buy a better, more expensive, alternative instead of this product. This is called up-selling. Both cross-selling and up-selling can be either personalized or non-personalized and will lead to selling more products. Another reason for using recommender systems is customer loyalty. By learning about customers, a value-added relationship can be created and personalized offers and settings can be provided. So, personalization can improve the satisfaction of both customer and company while interacting with each other on the e-commerce website.

In order to come to a personalized recommendation, information about customers has to be collected. This information can for example be mouse clicks, banners clicked, purchases, transactions, demographics, etc. For each visitor a personal profile can be created which can be used for recommendations to be based on. Today, lots of e-commerce websites use some kind of recommendation mechanism. These techniques are often based on either *item-to-item correlation* or *collaborative filtering*. Item-to-item correlation is a method for recommending products based on other products the customer has shown interest in. These products can for example be in the shopping cart or the customer may already own them. In the first case it is not necessary that the customer has identified himself, in the second case it is. Collaborative filtering, also called *people-to-people correlation* [6], recommends products to a user based on the correlation between him and other users. This method can either be manual, where the user has to rate products or automatic, where buying patterns or click-streams are used.

2.3 Software Architecture

In order to realize our system we need to make a software architecture. There are many different definitions of the term *software architecture*. One of the most famous definitions is given in the book *Software Architecture in Practice* [7] by Bass, Clements and Kazman:

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

2.3.1 Overview of Architectural Styles

In the past decades various architectural styles have been developed. All of them exist of components and connectors between those components. The differences between the various architectural styles depend on what the components and connectors look like and how they are combined. The resulting styles all have their advantages and disadvantages which makes each of them suitable for systems with particular characteristics. A software architecture can be described in terms of multiple views. The reason why multiple views are used is separation of concerns [8]. A view addresses the whole system with respect

to one or more concerns. Thus, looking at an architecture from different viewpoints makes it easier to separate the concerns of the stakeholders. For example, a stakeholder who only has to deal with the maintainability of the system can now just use the maintainability view which does not contain information about performance, security, etc. In this way he can work far more efficient than when the whole system is captured in just one single view. Below, a short survey of several famous architectural styles is given. Furthermore, their suitability for use with the sales support system is discussed.

The *Blackboard Style* is an architectural style that is based on one shared data repository, the blackboard, on which one or more components are operating. The components are usually not directly connected to each other, but are indirectly connected through the blackboard. The blackboard style is used most often for complex applications in research labs and the advantages do not scale down to simple problems [10]. For commercial applications there are better styles, which are described below.

As the name suggests, an architecture based on the *Pipes and Filters Style* exists out of pipes and filters. The filters are independent entities and are connected by the pipes. Each filter receives input from another filter through a pipe, performs some calculation on the data and then output it to another pipe. This pipe is connected to the input of another filter, etc. In this way a stream of data flows through the system from the first input to the last output. Filters only see the input data, not its source. Also the destination of the output data is not known by the filter. Each filter may have more than one input or output. Because all the filters are independent entities, they can easily be replaced by newer ones, so the maintainability is very good. Also reuse of the filters becomes very easy because of their independency. A great disadvantage of the pipes and filters style is that, because of its structure, it is more suitable for batch organization of processing rather than interactive applications [9, 11]. This makes the Pipes and Filters Style not very suitable for the sales support system, because of the importance of interactivity.

The *Client-Server Style* is a so called hierarchical style [11]. It has one or more servers providing services to one or more clients. Clients can only connect to the server at run-time. The server does not have to know the identity of the clients. There is a maximum number of clients that can be connected to the server at the same time. It is possible that the server performs all the calculations for the

clients. In this way, the client only has to do the presentation. The client is then called a thin client [12]. It is also possible that a part of the data processing is done by the client. The client then becomes a thick client. This architectural style looks very promising, because users can use their web browsers as clients and the sales support system will act as the server. Today, most commercial e-commerce systems that use personalization are based on some variation of this type of architecture [13].

Another hierarchical style is the *Layered Style*. It exists of different layers that provide services which can be used only by the layer directly above. Sometimes, when this mapping can not be made, layers are allowed to use services from other layers that are more than one level below them [9]. It is never possible to use services of higher-level layers. The pure layered style is particularly useful when data progresses through successive levels of abstraction [9]. However, in network-based systems the Layered Style is often combined with the Client-Server Style mentioned above. This results in the *Layered-Client-Server Style* [11]. An example of an architecture based on this Layered-Client-Server Style is the *three-tiered* architecture. In this architecture the server-side is split up in functional process logic and a data management part and the client takes care of the presentation [11]. There are also *two-tiered* and *multi-tiered* architectures. The three-tiered architecture might be suitable for the sales support system to be based on. The system can be split up in cleint, server and data layers. We will further expand this idea throughout this thesis.

Compared to the architectural styles described above, the *service-oriented architecture* (SOA) is relatively new. SOA exists of various independent, loosely coupled pieces of functionality, called services. Services interoperate based on a formal definition independent of the underlying platform or programming language. These properties makes SOA very interesting. Later on in this chapter we will give a more detailed description of SOA and other architectural styles that build on it.

The survey above makes clear that we have to look for an architectural solution in the field of the service-oriented architecture and the three-tiered architectural style.

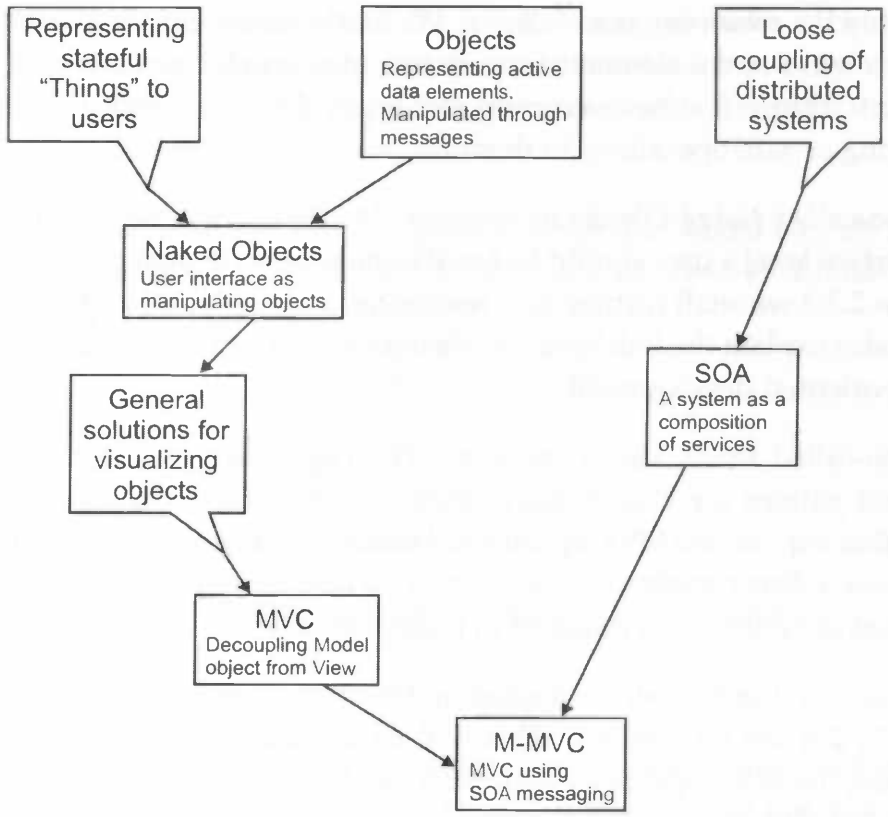

Figure 2.2: Reasoning of this background section

2.3.2 Reasoning about Required Solutions

Figure 2.2 shows the essential reasoning of the remainder of this section about software architectures. In callouts (squares with protruding triangles) certain requirements are stated. Rectangles represent (possible) solutions. Essentially, the figure represents the tension between two requirements that are not often combined in current software development, namely:

1. Manipulation of stateful elements. In eCRM stateful elements are manipulated, i.e. customers with as state a set of products that have been selected for them, or that they have selected.
2. The need for a distributed loosely coupled architecture. Especially an architecture where the graphical representation may be decoupled from the business logic.

Roughly the reasoning is as follows. We firstly follow the left part of the figure. To present stateful elements to users, the idea is to let users directly manipulate *objects*: things that have a certain identity and can be directly manipulated by applying certain operations to them.

The so-called *Naked Objects* movement [14, 15] adopts the idea that at the user-interface level a user should feel as if (s)he is directly manipulating objects. In section 2.3.3 we shall further give references and explain this. In this section we shall also explain the link between Naked Objects and the standard principle of object-oriented development.

The so-called *Model-View-Controller* (MVC) approach [16, 17, 18, 20, 19] is a standard pattern for visualizing objects in applications. In section 2.3.4 we shall further explain the MVC approach. However, the standard MVC approach is based on a direct method invocation in an object-oriented system, not on a distributed architecture as required in point 2 above.

To enable a distributed, decoupled architecture a *service-oriented architecture* (SOA) [21, 23] can be applied. This will be discussed in section 2.3.5 below. Combining the MVC approach with SOA can be done using the M-MVC principle [29, 30]. This will be discussed in section 2.3.7 below.

2.3.3 Naked Objects

Traditional systems are often based on a true business object model, but the core business objects are typically hidden from the user. The user usually has no indication that (s)he is working with an object model. The layered architecture shields the user from the structure of the underlying software and the user interface has been optimized to a particular set of tasks.

A concept for designing system architectures that works differently than the traditional ones is Naked Objects. Naked Objects is an approach for designing systems where the user interface is a one-to-one mapping of the business object model. By letting the user feel as if (s)he is directly viewing or manipulating objects from the model, the system becomes more expressive for the user. With Naked Objects, the interaction of the user with the system happens completely in the noun-verb style, i.e. objects can be selected after which the available methods can be called. Things that are common in most business systems, such as dialogue boxes, forms and message windows have been eliminated.

Using the Naked Objects Toolkit, the programmer can make an object model that has the ability to display itself and its business behavior directly to the user. In this way it takes less time to develop a user interface compared to traditional object oriented design and programming.

In our approach we will not completely adopt the Naked Objects concept. We will only use the idea that at the user-interface level a user should feel as if (s)he is directly manipulating objects.

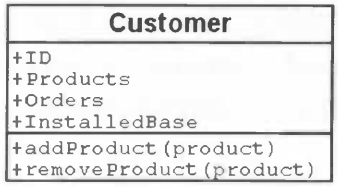

Figure 2.3: The customer represented as an object

In eCRM a customer can be represented as an object: the `Customer` object. Using this `Customer` object, the system keeps track of the state of each customer. The example `Customer` object, shown in figure 2.3, contains four attributes. The attributes have the following meaning:

id The unique identifier for the current customer.

products The set of products that is currently associated with this customer.

orders The set of running orders.

installedBase The set of products the customer already owns.

Furthermore, the methods `addProduct(...)` and `removeProduct(...)` can be used to respectively add or remove products associated with the customer.

When following the Naked Object concept, the user interface of the system displays the current state of the `Customer` object and the methods to change the state. For example, the user can change the state of the object by invoking the `addProduct(...)` method. After the action by the user, a product has been added to the `Customer` object and thus the state has been changed.

2.3.4 Model-View-Controller Pattern

The Model-View-Controller (MVC) pattern is used for separation of concerns. It has its origin in 1978 as the design solution to a particular problem [16, 17]. MVC separates the presentation (user interface) from the domain model (conceptual model of the system). Furthermore, it splits the user interface in a controller and a view. Although MVC is known as a design pattern, the term architectural pattern would be better [18]. This is because MVC is more related to architecture than typical design patterns. Figure 2.4 shows the relations between model, view and controller. The solid lines indicate a direct association, which are usually implemented as method invocations. The dashed lines indicate an indirect association, usually implemented as events.

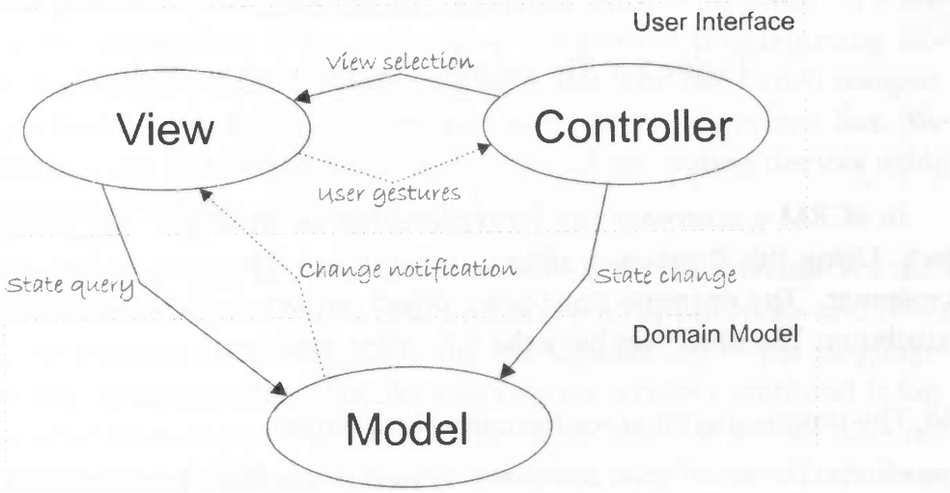

Figure 2.4: The Model-View-Controller

The *model* encapsulates data and rules for accessing and updating this data. The storage mechanism of the data is also encapsulated in the model, so how the data is stored is not specified. The *view* renders the model in such a way that it can be viewed to the user as part of the user interface. This can be done in many different ways, resulting in different forms of representation (e.g. tables and graphs). The role of the *controller* is to process events, which are typically user actions. The controller also may invoke changes on the model and view. Often, the controller and the view are combined in one user interface.

When the user clicks on a button, or interacts in another way with the user interface, the controller handles the input event. The controller accesses the model and, if necessary, changes its state according to the user's action. When the model has been changed, it notifies the view. The view fetches the new state and updates the user interface, which now waits for the following action performed by the user.

There may be lots of users (all with their own interface) interacting with the model at the same time and each user interface may contain different views of the model. The model has no direct knowledge of the existence of these views. However, when the model changes, the views have to be updated. Using the observer pattern [19], the model notifies the views when it has changed state. It remains the responsibility of the views to query the new state from the model and change their representation to preserve consistency.

Because of the CPU and network constraints in the past, software systems were built out of closely coupled components. This caused these systems to become very complex, resulting in lack of support for reusability and interoperability. MVC already has separate model, view and controller components, increasing reusability. According to respectively Moore's and Gilder's laws, computer processing performance and network bandwidth improves continuously. The standard MVC approach doesn't benefit much from the network bandwidth improvements because it is based on direct method invocation in an object-oriented system. Messaging over a network (e.g. the Internet) infrastructure becomes very interesting for model-view interaction because it benefits from both the CPU and network performance increases. Furthermore messages can be sent between systems from different platforms, which significantly improves the interoperability.

In the next section, the service-oriented architecture (SOA) is described. As stated earlier, SOA can be used to create a distributed architecture, that can be used in combination with MVC in order to fulfill the requirement stated in point 2 at the beginning of this section.

2.3.5 Service-oriented Architectures

In order to enable a distributed, decoupled architecture, a *service-oriented architecture* (SOA) can be applied. SOAs are used in the service-oriented computing

(SOC) paradigm that utilizes services as fundamental elements for developing applications [21]. In a SOA environment, resources in a network are made available as an interconnected set of services that are accessible through standard interfaces and messaging protocols. The definition of SOA, according to OASIS (the Organization for the Advancement of Structured Information Standards), is:

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

Services are self-describing, independent, loosely coupled pieces of functionality that perform specific functions. Services interoperate based on a formal definition independent of the underlying platform or programming language. Because of this property, service are very useful in distributed computing. Using a middleware infrastructure, such as the *Enterprise Service Bus*, distributed services can be invoked and a wide range of computing devices using various software platforms can be connected [22].

Services exist of an interface and an implementation. The interface is a declaration that describes how services communicate with applications and other services. The interface definition hides the implementation of the language-specific service. A service can be invoked by various service clients and is logically decoupled from any service caller. This means that the service does not have pre-knowledge of its clients and the clients don't need to have knowledge of how the service actually performs its tasks [23].

Services can either be *simple* or *composite*. Simple services are not built using other services, while composite services are. Composite services can be assembled from different simple or other composite services from one or more services providers. By composing services out of others, logic is divided into services, increasing the reusability. Services are most often designed to ignore the context in which they are used, so they are not context sensitive. The advantage of context insensitiveness is that these services can be reused in contexts not known at design time. Furthermore, reusing services is of a higher level than reusing objects since services can implement complete business processes. Besides good reusability, SOA is very flexible and maintainable because of the modularity.

Services are offered by service providers. A service provider provides service implementations, supplies the service descriptions and provides related technical and business support. Service descriptions are used to advertise the service capabilities, interface, behavior and quality. Service clients must be able to find the descriptions of the services and must be able to bind to them. These service clients can be applications within an enterprise or clients outside the enterprise.

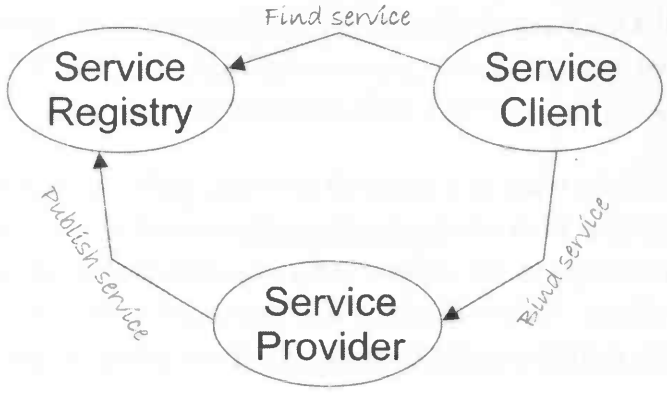

Figure 2.5: The service-oriented architecture (SOA)

As can be seen in figure 2.5, SOA is a relationship between three kinds of participants: the *service provider*, the *service registry* and the *service client* (the requestor of the service). These participants interact using the *publish*, *find* and *bind* operations. A service provider hosts an implementation of a given service. This service provider defines a description of the service and publishes it to a service registry which then publishes the description and makes it discoverable for service requestors. The service requestor retrieves the service description through a find operation from the service registry. Now, the service requestor uses the service description to bind with the service provider and invoke the service.

2.3.6 Web Services

According to the W3C a Web service is defined as *a software system designed to support interoperable Machine to Machine interaction over a network* [24]. However, in common usage the term *Web service* refers to those services that use SOAP-

formatted XML envelopes and have their interfaces described by WSDL. The most common specifications that define Web services are described below.

Interaction between Web services is realized by exchanging XML messages using SOAP. SOAP is an XML-based, extensible message envelope format. The service descriptions are expressed using *WSDL* (Web Services Description Language), which is also XML-based. WSDL describes the public interface, protocol bindings and message formats required to interact with a Web service. The *UDDI* (Universal Description, Discovery and Integration) standard is a XML-based directory service containing service descriptions (WSDL). Clients can use UDDI to locate candidate services and discover their details.

Web services can be used in a number of ways, including styles that use RPC and SOA [25]. The first Web service tools primary used RPC. Using RPC, a client sends a request message to the server, who immediately sends a response message back to the client. A drawback of this approach is that it is a synchronous way of interacting which results in more tight coupling of the components in the system.

Today, Web services have become one of the most important standards for realizing SOAs and has gained broad industry acceptance [23]. When using the SOA approach, the basic unit of communication is not an operation, as in RPC, but a message. This supports an asynchronous way of communicating, resulting in a looser coupling of the components.

2.3.7 Message-based MVC

Message-based MVC (M-MVC) [29] is a service-oriented architecture that builds on the MVC pattern. M-MVC is supposed to be a distributed version of the MVC pattern, using message-based interactions between model and view components. This enables long distance linkage between model and view. Furthermore, device and platform independency is realized.

M-MVC uses the *Publish/ Subscribe* paradigm for communication between the components. Publish/ Subscribe is an asynchronous messaging paradigm [26]. In a Publish/ Subscribe system, publishers post messages to an intermediary broker and subscribers register subscriptions with that broker. In a topic-based system, messages are published to *topics* which are hosted by a broker.

Subscribers will receive all messages published to the topics to which they are subscribed and all subscribers to a topic will receive the same messages. Publishers are loosely coupled to subscribers, and needn't even know of their existence.

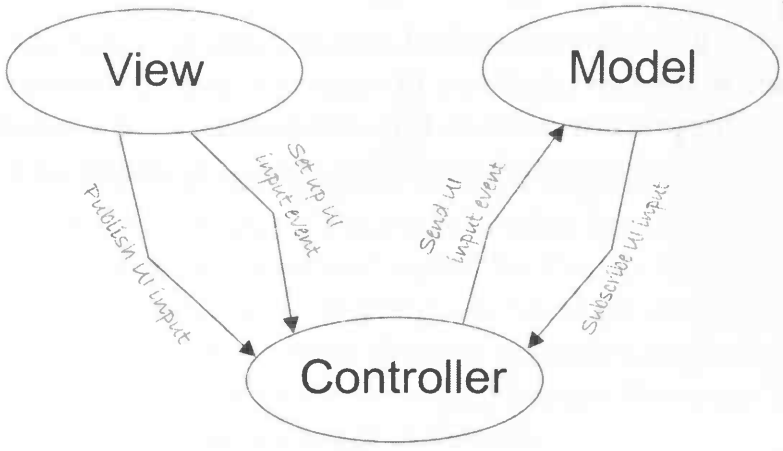

Figure 2.6: UI input event in the Message-based Publish/ Subscribe MVC model

Figure 2.6 illustrates how traditional MVC can be transformed in message-based MVC using the Publish/ Subscribe paradigm. It shows how the View component publishes its *User Input* event class to the appropriate topic at the Controller (the broker). The Model component subscribes to the topic. The View can now set up an User Input event to the Controller, which will send it to the Model. In order for the View to be able to update, the Model publishes its *Rendering* event class (this is not shown in figure 2.6). The View subscribes to the corresponding topic and is able to receive events from the Model (also through the Controller).

The events are handled using the *Web Services Eventing* specification (WS-E). WS-E is a Web services event system that defines how to construct an event-based message exchange pattern, which enables Web services to act as event sources for subscribers. It includes an event/listener pattern.

The M-MVC approach uses message-oriented middleware as the messaging infrastructure. Message-oriented middleware is inter-application communication software that generally relies on asynchronous message-passing. Other

middleware, like the Remote Procedure Call (RPC) pattern, uses a request/ response metaphor, which is synchronous.

2.3.8 Conclusion

After comparing the various described architectures, we conclude that the M-MVC approach is the best candidate to support a general decoupled architecture. However, for practical reasons, this architecture is not worked out in this thesis. The architecture that has been chosen instead, is described in chapter 3.

Chapter 3

Design

This chapter describes the design of the sales support system based on the requirements from chapter 1 and the theoretical foundation given in chapter 2. The system exists of a front-end, called the *Content Manager* and a back-end, called the *Scenario Manager*. The Scenario Manager contains the system's business logic and data. The Content Manager generates menus for the user interface using data received from the Scenario Manager. Users can interact with the sales support system using a range of devices.

The communication between the Content Manager and the Scenario Manager is based on message passing. The Scenario Manager receives request messages from the Content Manager and answers them with response messages. This is illustrated in figure 3.1. These messages only contain pure business data, so no information about the lay-out is included. In this way they are device and platform independent, and the coupling between components is small.

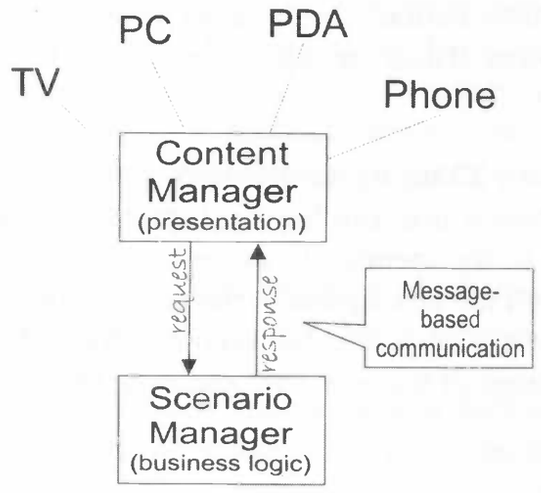

Figure 3.1: Main components and their connection

The emphasis in this chapter will be largely on the Scenario Manager component, including the databases to which it is connected and the message-based communication with the Content Manager. The design of the Content Manager component and the various device-specific interfaces as well as the design of the *Learning Machine*, which takes care of the self-learning aspect, have been outsourced by Atos Origin TUM to various students.

Two groups of about ten Computing Science students from the University of Groningen and the Vaxjö University in Sweden have taken care of the design of the *Content Manager* component as well as the interfaces for PC and TV devices. They have implemented their design as a prototype, which is described in chapter 4 as part of the prototype of the whole system.

The design of the *Learning Machine* component has been carried out by an Information Science student from the University of Groningen. A student from the Computing Science department of the University of Groningen has taken care of the design of the Telephone and PDA interfaces. These two interfaces as well as the *Learning Machine* component have not yet been implemented as a prototype at the moment this thesis was completed.

Section 3.1 describes the *Micro Funnel* concept, which is a sales concept for self-service using the Internet. It is based on the Sales Funnel mentioned in chapter 2 and will be implemented in the Scenario Manager component. One of the goals of the Micro Funnel concept is to personalize the sales process of the sales support system. It also provides a mechanism for keeping track of the state of users.

After that, in section 3.2 the menu structure of the system is described. This structure describes how a user can browse through the menus and also how products are added to the menus. In order to illustrate this, an example is worked out along with the description of the menu structure concept. This example is based on the prototype that has been developed for Atos Origin TUM. It describes how to support the user with choosing from a set of products (P1, ..., P9).

Finally, the architecture of the sales support system is described in section 3.3. This includes the message-based communication between the two main components: the Scenario Manager and the Content Manager. The various mas-

sage types are explained and use cases are given to show how they are used to synchronize the two components when the user's state changes.

3.1 The Micro Funnel Concept

In chapter 2 we already gave a short introduction to the traditional sales process and the Sales Funnel. We will now introduce a sales process concept that is based on the Sales Funnel. The traditional Sales Funnel is designed to be used for sales processes with personal contact between customer and seller. In our situation we are dealing with self-service. In stead of personal contact with a sales person, the customer has to use the Internet to interact with the sales support system in order to be served. Furthermore, the traditional Sales Funnel is normally used in long sales processes instead of our relatively short process. We have designed a Sales Funnel that is more suitable for self-service through the Internet infrastructure. Because the accent in our approach is on a short sales process, we call our funnel: *Micro Funnel* (μF).

In figure 3.2 the Micro Funnel is shown. The time it takes a customer to complete all steps, including the transaction, can vary from about one minute to an hour. At the top of the Micro Funnel are all the visitors of the website. There are people who are just surfing around and people who already know what they want. At the bottom of the funnel, are the people who have bought one or more products and have completed the transaction. The Micro Funnel exists of a fixed number of steps. Depending on the demand of the customer, he either processes all the steps or only a few. Furthermore, it is possible to go back to a previous step (e.g. to make corrections) and to resume the sales process after interruption. How the customer proceeds the steps, is directed by the Scenario Manager component.

The numbers in figure 3.2 represent the steps which the customer has to take in order to move through the Micro Funnel. These steps include:

1. **Identify actual customer's need.** The reason why the customer visits the site is discovered. The customer has to choose an option from the main menu.
2. **Determine main choice.** After the customer has chosen an appropriate option from the main menu, he will be taken to the right sub menu. In this menu he can compare products or services that are selected according to

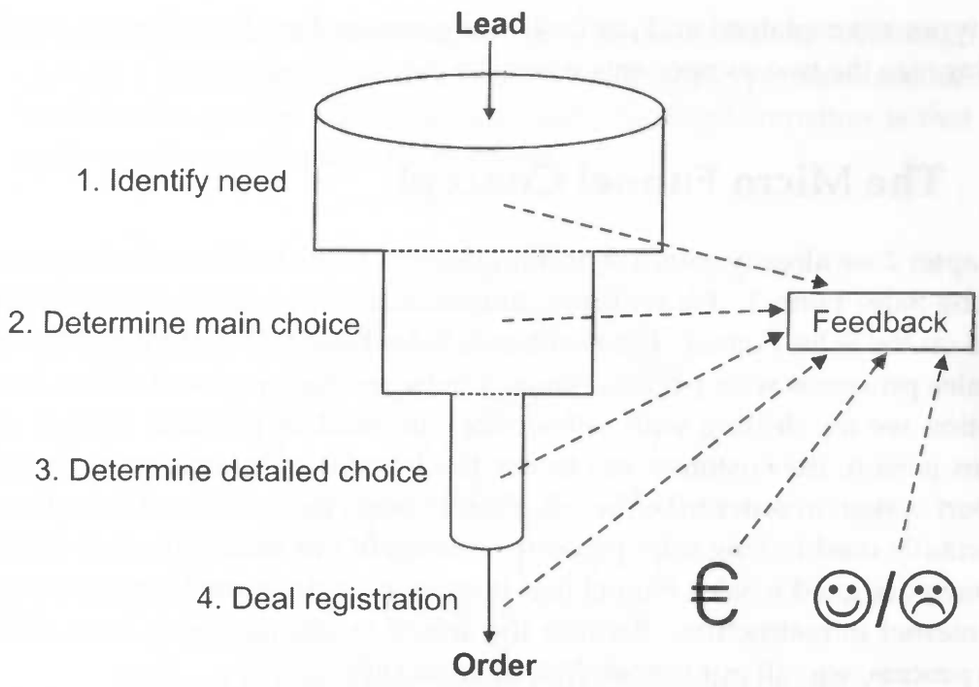

Figure 3.2: The Micro Funnel

his choice in the previous step. Also, using the customer's profile, recommendations for up-selling can be made. In this way advanced products are more emphasized than basic ones if appropriate when looking at his profile.

3. **Determine detailed choice.** In this step the customer enters a sub sub menu. This menu contains the actual product selected in the previous step. In this menu it is also possible to offer complementing products like supplies and add-ons to the customer (cross-selling).
4. **Deal registration.** The software for completing the deal is started. When finished, the customer returns to the main menu.

The dotted lines indicate the various sources from which feedback is received. The feedback can be used afterwards to optimize the Micro Funnel. The feedback includes:

- Results from the steps in the Micro Funnel
- Details of all orders

- Customer satisfaction related to the delivered products and services
- Payment details related to the delivered products and services

3.1.1 Tracking the users' state

For each customer that enters the Micro Funnel, a state being is kept. This state indicates how much information the customer has provided about himself at the moment. While the customer moves through the Micro Funnel, this state may change. There are three possible states, called *domains*, in which a customer can be, namely:

G-domain Generic selection of the portfolio. The customer is anonymous and content is the same for everyone in this domain.

P-domain Profile-based selection of the portfolio. The content is based on a profile constructed with data provided by the customer. In this state, the customer remains anonymous: only some of his characteristics are used.

I-domain Individual selection of the portfolio. The content is based on the customer's individual personal data. The customer has identified himself and is no longer anonymous.

Because of the names we made up for the three domains, we call the resulting model: *GPI domain model*. The content shown to the customer is based on the domain he is in. It is possible to link the states in the GPI domain model to the steps in the Micro Funnel, but this relation is not fixed. Nevertheless, when the customer enters the website he is in the G-domain and he can not proceed to the checkout until he is in the I-domain.

Figure 3.3 summarizes the possible transitions between the three states in the GPI domain model. As can be seen, there are four types of transitions which we call *functions* between the states, namely:

Profile Takes the customer from the G-domain to the P-domain. In order to do this, the customer has to fill in a profiling form.

Login Takes the customer to the I-domain. This function requires some kind of authentication, e.g. a user name/ password combination.

Unprofile Takes the customer back from the P-domain to the G-domain.

Logout Takes the customer from the I-domain to the P-domain. Although the customer has logged out, there still may be some personalized advice based on his profile. The difference with the *Profile* function is that the customer does not have to fill in a form.

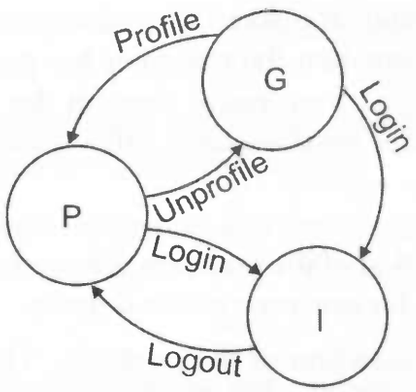

Figure 3.3: Transitions within the GPI domain model

3.1.2 Guiding using Scenarios

The customer is guided through the Micro Funnel using *scenarios*. Scenarios are some kind of use cases managed by the Scenario Manager component. Depending on the customer's need the right scenario is started in the first step of the Micro Funnel. Scenarios determine which actions have to be taken and which products have to be shown when a customer selects a particular menu. When the customer makes a specific choice in one of the menus, the Scenario Manager defines which step has to be taken according to the current scenario. This can be:

- Display another menu
- Advice the customer to login
- Force the customer to login
- Advice the customer to profile
- Force the customer to profile

The first option, *Display another menu*, is used when the customer is already authorized to enter the particular menu. Otherwise, the Scenario Manager determines which of the other options applies for the menu. As a result, either a log in screen or a profiling form is presented to the customer before he can proceed entering the menu. In order to keep track of the customers, the following information is kept with the scenario that the customer follows:

Session ID Unique identifier for the current session.

The time when the scenario was started Logged for analytical purposes.

The time when the last user input has been received Used to generate a time-out when no input has been received from the user for some time.

The current menu The last menu sent by the Scenario Manager

The last menu/ product choice Has the same value as *current menu* when the menu has been entered successfully. Otherwise it contains a product or the menu for which logging in or profiling is needed in order to enter.

The state in the GPI domain model Used in order to determine which personal data may be used and to verify if the user is authorized to enter selected menus.

3.2 Menus and Products

As stated in section 3.1 about the Micro Funnel concept, the content of all the menu screens (including the main menu) is based on the amount of personal data that is known about the customer. It is the role of the Scenario Manager component to combine data from various sources in order to construct the menu content, including personalized offers. The amount of data that can be used is based on the customer's state in the GPI domain model. The Scenario Manager component uses the following input for constructing the content of menu screens:

- The menu option chosen by the customer.
- The desire or the problem of the customer (the context of the visit). This can be determined by the customer's choice at the main menu.
- The current menu the customer is in. This may indicate in which direction the customer looks for a specific solution.

- Available personal data of the customer (customer picture). The amount of data that will be used depends on the amount of personal data the customer wants to be used for the advice. This corresponds to the customer's current state in the GPI domain model. Further personal data that may be used includes:
  - Running business between customer and supplier (corporate customer picture);
  - Customer segments (e.g. social demographic, value-based or attitude-based) to which the customer belongs.
- The complete portfolio of the supplier (corporate assortment). This are the products and services from which the Scenario Manager has to make its choice.

When the customer has selected an option in one of the menu screens and the Scenario Manager component has processed the data, it gives as output a new menu screen, a log in screen or a profiling form. In order to illustrate the menu structure and how products are determined we will introduce an example which will be used throughout this section.

3.2.1 Main Example

The main example illustrates the sales support system's menu structure, how products are added to the various menu screens and the profiling concept . It is based on the prototype that has been developed for Atos Origin TUM together with this thesis. The prototype is based on the e-commerce website of one of the largest telecom operators in the Netherlands. It will be discussed in chapter 4. The following menus (M0-M4) and products (P1-P9) will be used in the example:

M0 *Main Menu* Home page where the user enters the website.

M1 *Home Telephony* Products and services related to fixed telephony.

M2 *Mobile Telephony* Products and services related to mobile telephony.

M3 *Broadband Internet* Products and services related to Internet Access.

M4 *Do the Telecom Scan* Personalized advise on products and services based on a profiling form.

P1 *Call Buget*, home phone subscription

P2 *Call Basic*, home phone subscription

P3 *Call Without Caring*, home phone subscription

P4 *Mobile 100*, mobile phone subscription

P5 *Mobile 200*, mobile phone subscription

P6 *Mobile 500*, mobile phone subscription

P7 *ADSL Lite*, Internet access subscription

P8 *ADSL Basic*, Internet access subscription

P9 *ADSL Extra*, Internet access subscription

We will now continue with explaining the menu structure concept while expanding the main example for illustration.

3.2.2 Menu Structure

In our menu structure concept, a menu is represented as a list. This list contains all the data that is necessary to generate a menu screen, which can be viewed by the user. Each menu (screen) may contain one or more links to other menus and/or one or more products. The menu structure and the menus are fixed, except for the listed products. Each menu has its own id number which can be used to link between this and another menu. The menus are divided into three levels. Each level corresponds to a step in the Micro Funnel:

Level 1 Step 1 in the Micro Funnel: *Identify actual customer's needs.*

Level 2 Step 2 in the Micro Funnel: *Determine main choice.*

Level 3 Step 3 in the Micro Funnel: *Determine detail choice.*

The Scenario Manager directs how the customer is guided through the different menus, i.e. which options are available in the menus. The structure of the menu levels is shown in figure 3.4. On level 1 there is just one menu screen, the *Main Menu*, which is supposed to be a portal where the user enters the website. On level 2 are sub menus which form the product categories that are available from the main menu. On level 3 are sub sub menus which we call *product nodes*. A product node contains information about the product in question. Besides

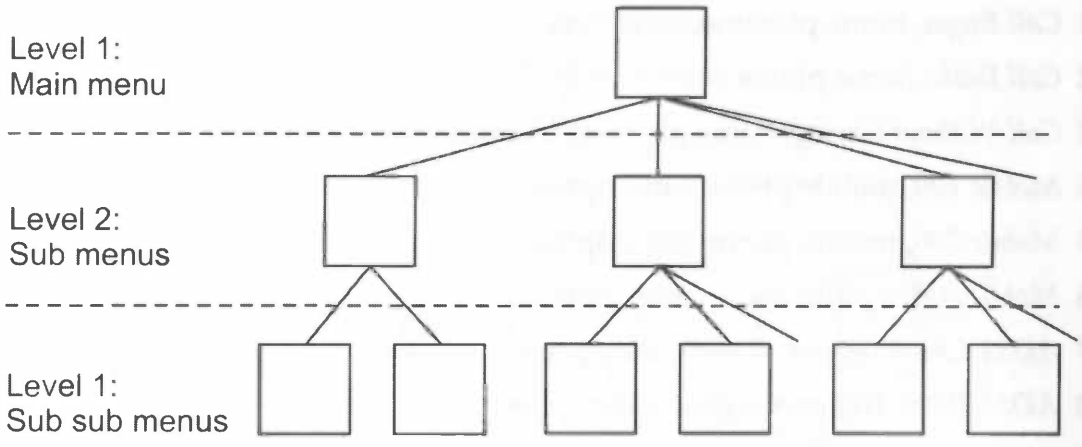

Figure 3.4: Structure of the menu levels

information about the current product there may also be cross selling and up selling activities by the presence of links to other product nodes.

Product nodes also contain buttons like *Add to shopping cart* and *Buy this product directly*. The shopping cart is kept by the Content Manager component. When the customer decides to buy the items in his shopping cart, all the items are sent to the ordering software at once.

When the term *product* is used in this text as an item in a menu, we actually mean *product node*. So, when a menu contains a certain product, it contains the product node. This product (node) contains information about the physical product.

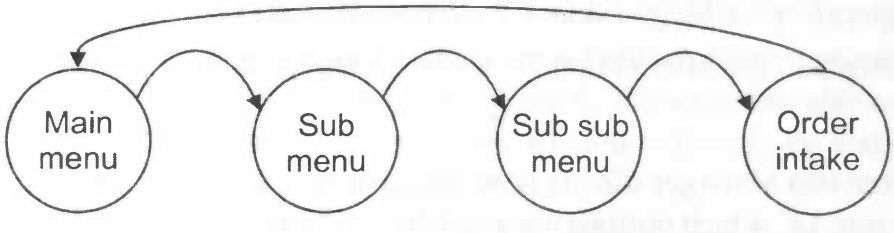

Figure 3.5: Standard path for browsing through the menu screens

Figure 3.5 shows the standard path for browsing through the menus. When following this standard path, it is presumed that the customer enters the website

in its main menu and selects an option which brings him to a sub menu. In this sub menu an option is selected which brings him to the corresponding sub sub menu, from which he proceeds to the order intake software. Finally, when the order has been registered, he will be sent to the main menu again.

The menu structure is not necessarily hierarchical: it is allowed that on menu can be reached by more than one menu from a higher level. Furthermore, for some options from the main menu, the sub menu, sub sub menu or both may not be used. There may for example be a special offer which sends the customer directly to the order intake or to the product node for some cross selling activities. It may also be possible to go from a menu to the same or another menu on the same level.

Although the menu structure is fixed, some items may be filtered away if they do not apply for the current customer. This can be because the offered product or service is not available for the customer. It is also possible that the customer has to be logged in or profiled in order to see some items of the menu. A menu structure exists of a number of menus and links by which users can get from one menu to another. Such a menu structure is represented as directed graph.

Figure 3.6 shows how the menus (M1-M4) and products (P1-P9) of the main example can be structured as such a directed graph. At level one there is one menu: the *Main Menu*. On the second level there are four sub menus, namely: *Home Telephony*, *Mobile Telephony*, *Broadband Internet* and *Telecom Scan*. On level three there are nine product nodes, which contain the products *P1* to *P9*.

The vertices in the graph represent menus. Edges represent the menu items (i.e. links) which point to other menus. Each vertex contains some elements. The first element in the vertex is the menu id number, which is used to identify the different menus. The second element represents the state in the GPI domain model (G, P or I) which is required (m) or advised (a) to enter the menu in question.

As can be seen all the menus, except *M4*, are labeled with a *G*. So, this means that these menus may be entered without profiling or logging in. Menu *M4*, which contains the Telecom Scan, is labeled with *Pm*, meaning that it is mandatory to fill in a profiling form to enter the menu.

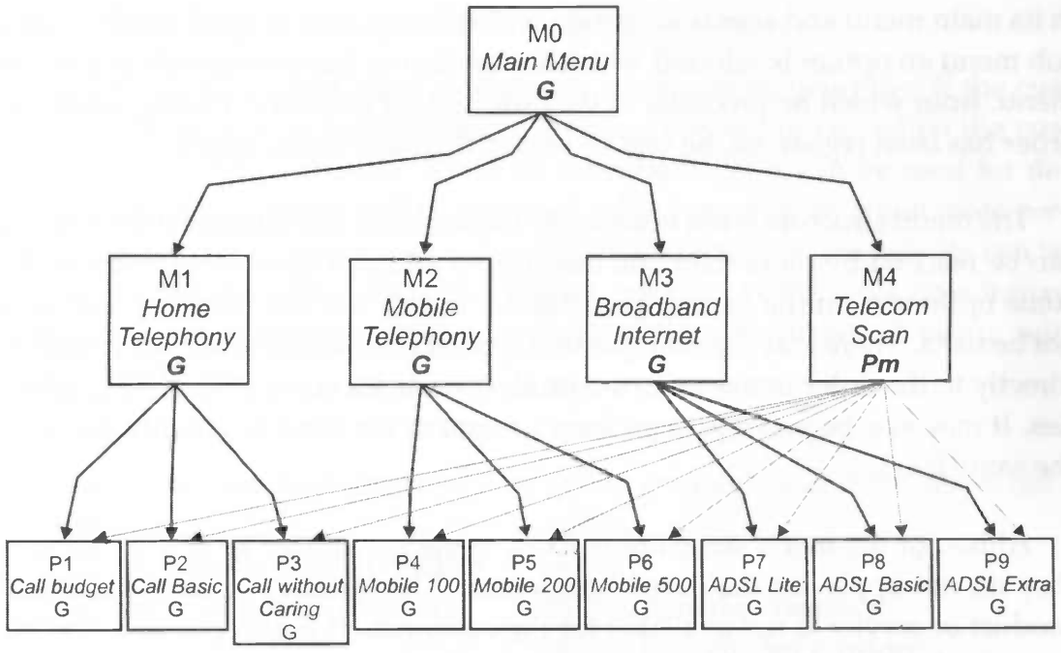

Figure 3.6: Menu structure represented as a graph

When the user clicks on a menu item directing to another menu vertex containing a P or an I label, and he is not already in the right domain of the GPI model, respectively a profiling or login screen is returned first. Otherwise the menu is entered without intervention. If it was advised (lower case *a* in the vertex) to login or profile, the menu is entered anyway. Only the content of the menu may be less personalized if the user did not provide the information asked for. When it was required (or mandatory, lower case *m* in the vertex) to login or profile, the menu will only be entered if respectively the login information provided is correct or the profiling form is filled. Note that some of the fields of a profiling form can always be optional. An edge can also be labeled with respectively a P or an I, meaning that the link will only be visible to the user if he has already profiled or logged in.

3.2.3 Products

Besides menus, there are products which have to be put in the menus as personalized as possible. In order to achieve this, we have defined the following strategy. First, we define a list of default products for each menu. These are the

products which the user sees when he is in the G-domain. Each menu-product pair also has a *priority* attribute, indicating which products are more important for a particular menu than others. Products can be sorted based on their priority attribute, determining the standard order in which products will be shown in a particular menu.

In order to realize this concept, we will use a table. We have chosen for this approach because tables can be stored in a database, after which they can be queried easily. We have defined a table which contains menus as rows, products as columns and priorities as values. A priority may be any number from 0 to 999, with 0 the lowest and 999 the highest priority. Values outside this range will be set to either 0 or 999. A priority of 0 means that the product should not be shown in the menu. Any higher value indicates that there is some relation between the menu and the product. An example table containing menus, products and their priorities is shown in table 3.1. In order to make the table easier to read, the priorities with value 0 have been leaved out.

	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**
M0									
M1	101	102	103						
M2				101	102	103			
M3							101	102	103
M4	100	100	100	100	100	100	100	100	100

Table 3.1: Menus (M0-M4), products (P1-P9) and their corresponding priorities.

The example shows that menu M0 contains no products at all. Menu M1 contains three products: P1, P2 and P3. Menu M2 contains the products P4, P5 and P6. Products P7, P8 and P9 can be found in menu M3. Menu M4, which contains the Telecom Scan mentioned earlier, contains all the products with priorities set to 100. This makes it possible to determine the eventual order of the products in this menu using the profiling form. How this work, is described below.

3.2.4 Profiling

The default products and their order in the menu can be overruled if the customer has logged in or profiled before entering the menu, i.e. when the customer is in the P or I domain. Table 3.1 described above, can be used for determining which products will be removed, added or replaced when more personal data about the current user is known. Products with a low priority can be dropped earlier than the ones with a higher priority. At the same time it may be possible that a product which was not visible in the standard menu, will become visible now.

1. What is your household situation?
   (a) Single
   (b) Living together without children
   (c) Living together with children

2. What is your age?
   (a) Under 20 years
   (b) 20-40 years
   (c) 40-60 years
   (d) Over 60 years

3. Do you have a fixed telephone connection?
   (a) No
   (b) Yes, from this company
   (c) Yes, from another provider

4. Do one or more household members call mobile?
   (a) No
   (b) Yes, from this company
   (c) Yes, from another provider

5. Do you have Internet through a fixed connection?
   (a) No
   (b) Yes, using ADSL from this company
   (c) Yes, using a dialup connection from this company
   (d) Yes, using ADSL from another provider

6. I would like a solution that is
   (a) the most complete
   (b) the most efficient
   (c) the cheapest

Table 3.2: Example profiling form

Table 3.2 shows a sample profiling form. This example deals with what is called the *Telecom Scan*. Using the Telecom Scan users are able to get an indication which products are useful for them based on some of their characteristics.

The characteristics used are *household situation* and *age*. Furthermore the advise is based on products that the user already might own and an indication of the price range in which he is looking.

Once a customer has profiled himself, the priorities and thus the order of products for a particular menu may change. This depends on table 3.1 with priorities mentioned above, and on another concept which we call the *fit factor*. These fit factors indicate how much a particular product matches with the current user's profile. For now, we assume that such a profile is constructed using a profiling form containing several multiple choice questions.

For this concept we introduce another table which contains profile answers as rows, products as columns and fit factors as values. A profile answer is one of the possible answers to a particular question from the profiling form. So a profile question with three possible answers will result in three table rows. The products are the same as in the priority table. The fit factors are values by which the product's priorities are multiplied. A fit factor can be any number larger than or equal to 0. When a priority is multiplied with a fit factor of 0, the priority becomes 0 and remains 0 since multiplication with 0 always results in 0. Any other value of a fit factor smaller than 1 will result in a lower priority. A fit factor equal to 1 is neutral and preserves the current priority. Finally, any fit factor larger than 1 increases the priority.

The new priorities of products in a particular menu are calculated by multiplying the rows from the fit factor table corresponding to the answers provided by the user with the row of the menu in question from the priority table. This results in a menu containing products based on the standard order, corrected by the provided profile answers. Note that it is very important to choose the right values for both the priorities and the fit factors in order to get good results. The prototype which has been developed and is described in chapter 4 can be used to experiment with these values. An example is given in table 3.3.

Table 3.3 assigns fit factors to the answers of the profiling form given in table 3.2. If we want to enter the *Telecom Scan* menu, we can give, for example, the following answers to the profile questions from table 3.2:

```
1. a      4. b
2. b      5. d
3. a      6. b
```

	P1	P2	P3	P4	P5	P6	P7	P8	P9
1a	0.5	0.4	0.25	1.25	2	1.25	1.25	2	1
1b	0.75	1.5	0.75	0.5	1	2	0.5	1	2
1c	0.75	1.5	2	0.5	1	2	0.5	1	2
2a	0.25	0.5	0.25	0.5	1	2	0.5	1	2
2b	0.75	1.25	1	0.5	1	2	1	1.5	2
2c	1	1.5	0.75	1.25	1.5	1	1.5	1	0.75
2d	1	2	1	1	0.5	0.25	0.75	0.5	0.25
3a	1	1	1	1	1.25	1.5	1	1	1
3b	0	0	0	1	1	0.75	1	1	1
3c	1	1	1	1	1	0.75	1	1	1
4a	1.5	1.5	1.5	1	1	1	0.75	0.75	0.75
4b	0.75	0.75	0.5	0	0	0	1	1	1
4c	0.75	0.75	0.5	1	1	1	1	1	1
5a	1	1	1	1	1	1	2	2	2
5b	1	1	1	1	1	1	0	0	0
5c	0.5	0.5	0.5	1	1	1	2	2	2
5d	1	1	1	1	1	1	2	2	2
6a	0.25	1	4	0.25	1	4	0.25	1	4
6b	1	1	1	1	1	1	1	1	1
6c	4	1	0.25	4	1	0.25	4	1	0.25

Table 3.3: Products (P1-P9), profile items (1a-6c) and their corresponding fit factors

When filling in these answers we represent a person who is a single of 20-40 years old with no fixed telephone connection, a mobile phone subscription of this company and an ADSL subscription from another company that is looking for the most efficient solution.

If we now take row *M4* of table 3.1 (the Telecom Scan menu), and multiply this with the multiplication of rows *1a, 2b, 3a, 4b, 5d and 6b* (the answers given) we get the calculation given in table 3.4. The results show that *P4, P5* and *P6* all have a priority of 0, meaning that these products should not be shown in the menu. This is because this are the mobile phone subscriptions, and we already have a subscription with the company. Also the products *P1, P2* and *P3* do not score very high compared to the normal value of 100. This means that the

	P1	P2	P3	P4	P5	P6	P7	P8	P9
1a	0.5	0.4	0.25	1.25	2	1.25	1.25	2	1
2b	0.75	1.25	1	0.5	1	2	1	1.5	2
3a	1	1	1	1	1.25	1.5	1	1	1
4b	0.75	0.75	0.5	0	0	0	1	1	1
5d	1	1	1	1	1	1	2	2	2
6b	1	1	1	1	1	1	1	1	1
multiplied	0.28	0.38	0.13	0.00	0.00	0.00	2.50	6.00	4.00
M4	100	100	100	100	100	100	100	100	100
multiplied	28	38	13	0	0	0	250	600	400

Table 3.4: Calculation of priorities for products (P1-P9), in menu M4 using given profile answers (1a-6b)

products related to home telephony do not have a high correlation with our profile. Maybe *P2* can be shown somewhere in the menu if there is enough place since it has the highest priority. Products that have high scores are *P7*, *P8* and *P9*. They all have priorities higher than 100. This means that the ADSL subscriptions should have a prominent place in the menu. Since *P8* has the highest priority, it should have some special attention. It may for example be shown as a *special offer* to the customer.

When a user logs in, his available profiling data is stored. In this way personalized menus can be displayed when he returns later on. When the customer logs out, he will move from the I to the P domain. The stored profiling data can still be used until the user chooses to unprofile. Note that it is always the responsibility of the Content Manager how the products are visualized to the user. The Scenario Manager only makes the calculations and sends the results.

3.3 Architecture

In this section we will extend the architectural concept given in figure 3.1 at the beginning of this chapter in order to create the specific architecture for Atos Origin TUM's sales support system. As said before, the architecture will consist of two main components:

Scenario Manager Around this component the Model part of the system is built. It contains the business logic of the sales support system.

Content Manager This is the main component of the View part of the system. It takes care of the presentation of data to the user's devices. It also includes Controller functionality for handling the user's input.

The complete architecture of the sales support system, describing the decomposition of the system into components and their relations, is shown in figure 3.7.

We will first start with a short description of all the components from figure 3.7. First the components that are included in the *View*:

Content Manager component Manages the various concurrent user sessions and turns the unformatted data received from the Scenario Manager component into a attractive webpage that can be presented to the users. The Content Manager component includes a database for storing data such as session and product information.

PC, TV, PDA and Telephone interface components Interfaces between the Content Manager component and the various corresponding devices. Each interface has its own database which contains device-specific data. Such data may include HTML frameworks for displaying web pages, product descriptions or multimedia files for users with a broadband connection.

The following components are part of the *Model*:

Scenario Manager component Takes care of choosing and following the right scenarios in order to guide the customer through the Micro Funnel. The Scenario Manager component receives requests from the Content Manager component. The requests are processed, logged and responded.

Logging database component Stores log data from the Scenario Manager component which can later on be used by the Learning Machine for optimizing the sales process or for other (analytical) purposes.

Scenarios database component Stores all the available scenarios.

Portfolio database component Stores the company's portfolio and its menu structuring.

Profiles database component Stores customer profiles.

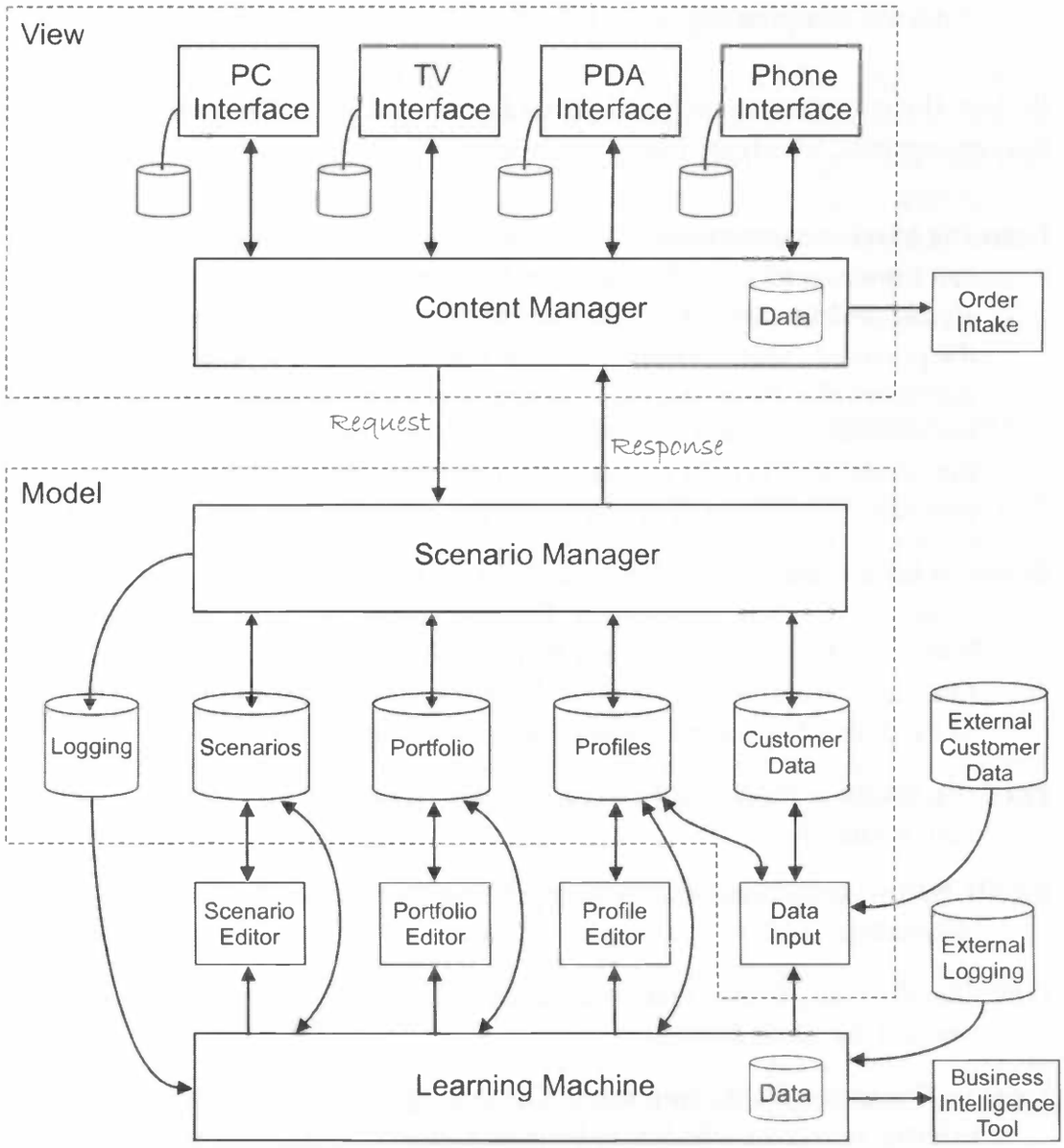

Figure 3.7: Architecture of the sales support system

Customer data database component Stores collected data about individual customers.

Data Input component Receives data from various sources, processes it, and outputs it to either the Profiles database component or the Customer Data

database component.

Besides the components included in the Model and View there are some external components, which are described below:

Learning Machine component Receives log data from the Logging database component and from the External Logging component. This data is analyzed and the results can be used to improve the scenarios, profiles and the portfolio. This can either be done manually, using one of the editors, or automatically. Note that the Learning Machine component is not included in the *Model*. The idea is that it is an external system that is decoupled from the Model and is only connected when data is needed from the Logging database component or when one of the databases has to be updated.

Scenario Editor component Editor that can be used for manually updating the Scenarios database component. Existing scenarios can be fetched from the Scenarios database component and combined with data from the Learning Machine. When the operator has created a new scenario it can be stored in the Scenarios database component and is ready for use.

Portfolio Editor component Editor for updating the Portfolio database component manually.

Profile Editor component Editor for updating the Profiles database component manually.

Order Intake component Takes care of closing the deal once the customer has pressed the *Order* button.

External Customer Data component Contains individual customer data from external resources which may be used to update the Customer Data database component.

External logging component External loggings which may be useful for improving the sales process by the Learning Machine component. This may include log data about order intakes, payment details and customer satisfaction.

Business Intelligence Tool component A tool, such as Business Objects [32], for creating various reports of business data.

3.3.1 Messaging Between the Main Components

In the theoretical background of chapter 2 we stated that the communication between the *Model* and *View* components should be message-based. In the architecture shown in figure 3.7 can be seen that the two components that perform this communication are the *Content Manager* and the *Scenario Manager*. In fact, the Content Manager component sends request messages to the Scenario Manger component. The Scenario Manger component processes the requests and sends a response message back to the Content Manager component for each received request message.

We have defined two groups of messages. The messages belonging to the first group are used by the Content Manager component as requests. The second group consists of messages which are returned by the Scenario Manager component as a response. The message types below are used for requests by the Content Manager.

MenuChoice Request for a particular menu. This request is sent when a user selects a link to one of the menus at the user interface.

LoginAction Request for a personalized version of the current menu. This request is sent after the user has succesfully authenticated at the Content Manager. The user's identification is included in the request and is used for personalizing the menu.

LogoutAction Request for a non-personalized version of the current menu. This request is sent after the user has logged out at the Content Manager.

ProfileAction the customer's answers to a particular form are sent. The customer moves to the P-domain.

UnProfile Request for a general version of the current menu. This request is sent after the user has indicated at the Content Manager that no profiling data has to be used for the content of the menus.

The LoginAction, LogoutAction, ProfileAction and UnProfileAction message types are directly related to respectively the Login, LogoutProfile and Unprofile functions described in section 3.1.1 about the GPI Domain Model. When the Scenario Manager receives one of these messages, the corresponding functions are called in order to take the current user to another domain in the model.

The Scenario Manager component can answer received requests with one of the following message types:

MenuRequest Request to show a particular menu. This request includes a list of products to be shown together with their priorities as defined in section 3.2.

LoginRequest Request to let the customer login. This request is sent when a user want to enter a menu which requires (or sometimes advises) him to login.

ProfileRequest Request to let the customer fill in a profile form. This request includes the profiling form.

All the request and response messages are defined as classes containing attributes which form the content of the messages. The super class of response messages is called ResponseTransaction. For request messages we call the super class RequestTransaction. Figure 3.8 shows a UML class diagram with the RequestTransaction super class and its sub classes. Figure 3.9 shows the same generalization structure for the ResponseTransaction super class.

As can be seen in figure 3.8 and 3.9, the classes ProfileAction and ProfileRequest have a whole-part relation with respectively the classes ProfileAnswer and ProfileQuestion. So, a ProfileRequest object contains one or more NumberProfileQuestion, SingleChoiceProfileQuestion or MultipleChoiceProfileQuestion objects. ProfileQuestion and ChoiceProfileQuestion are abstract classes of which no objects can be created. This structure, as well as the structure under the ProfileAction class is used for creating messages containing flexible profile forms.

The profiling concept in section 3.2.4 only supports profile forms in which an answer to a particular question exists of one or more options in a list of answers. A message object containing such a profiling form is constructed as follows:

1. A ProfileRequest object is created. If profiling is mandatory, the attribute `mandatory`is set to `true`, otherwise to `false`.
2. For each question in the profile form, one of the (non-abstract) subclasses of ProfileQuestion is created, depending on the type of question, and included in the ProfileRequest object as attributes (or as one array attribute containing all included classes). Each of the objects contains an attribute called `question`, containing the question in text format and an attribute called required, indicating if this question should be answered or may be skipped. If the question should be answered with a number, a NumberProfileQuestion object is created, containing no extra attributes. If the

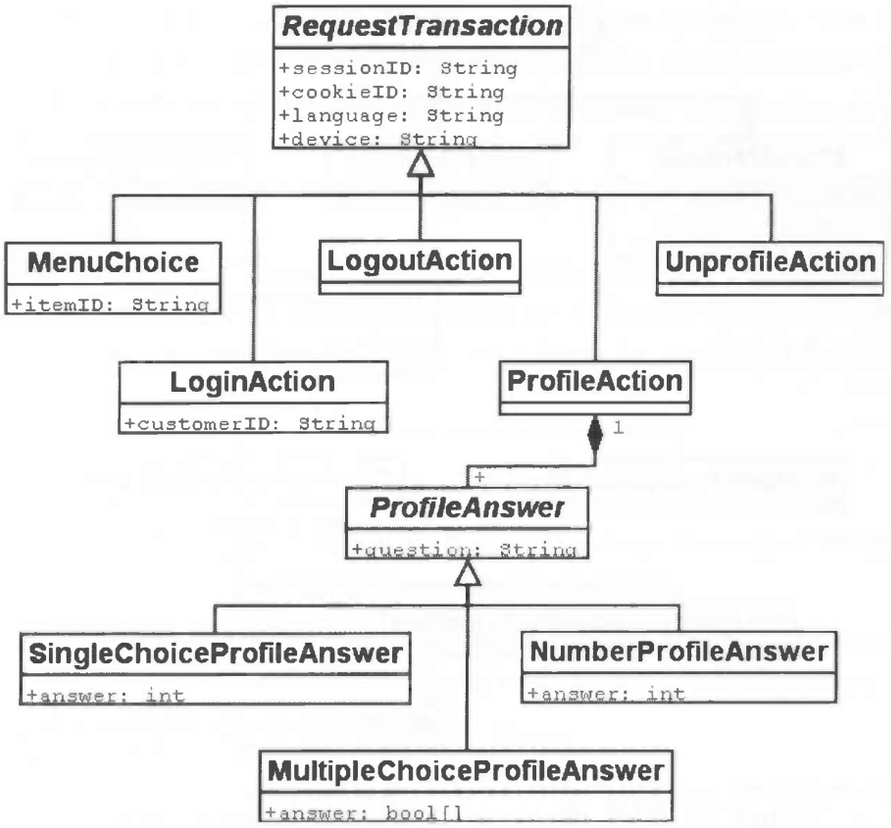

Figure 3.8: UML class diagram for request messages

question should be answered with exactly one option from a list of possible answers, a SingleChoiceProfileQuestion object is created, containing an attribute called `answer` which is an array of strings containing all the possible answers. This array is also included when a MultipleChoiceProfileQuestion object is created, meaning that the more answers per question can be chosen from the list.

3. If all the questions are added to the ProfileRequest object, it can be serialized and send to the Content Manager.

After the Content Manager has received the message, and the user has correctly filled in the profiling form, a ProfileAction object will be sent to the Scenario Manager. This object contains the answers provided by the user. Such a ProfileAction object contains one or more objects of the subclasses of the abstract class ProfileAnswer, depending on the type of question (just as described

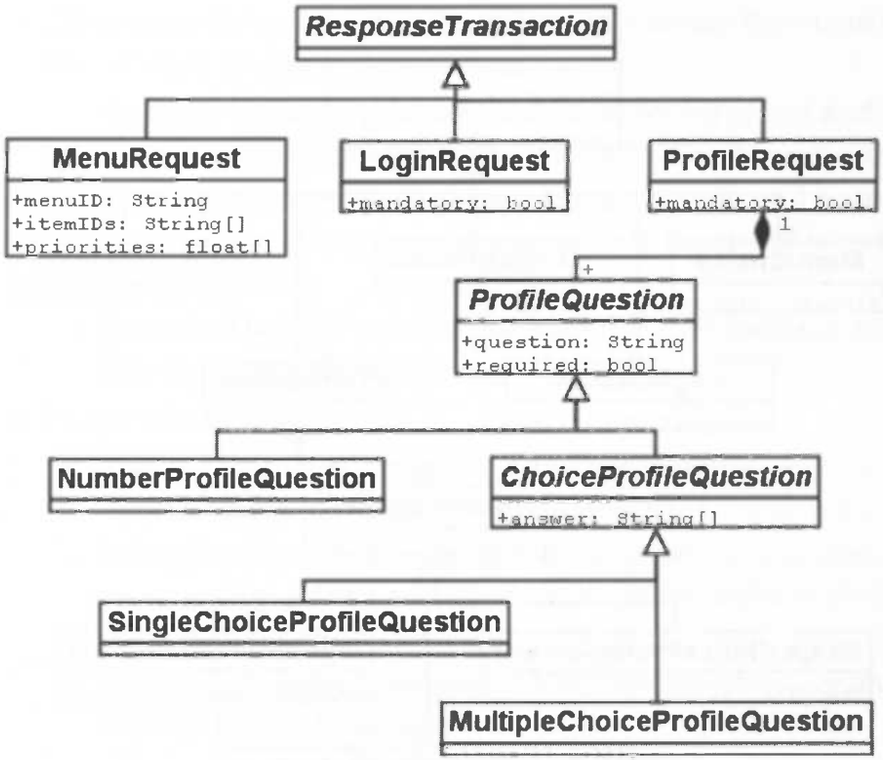

Figure 3.9: UML class diagram for response messages

above). Each object contains a string with the original question as attribute. Furthermore, the NumberProfileAnswer object contains the value that the user has answered. The SingleChoiceProfileAnswer object also contains a number, indicating which answer was chosen. Finally the MultipleChoiceProfileAnswer object contains an array of Boolean values, which indicates for each possible answer if it was chosen (`true`) or not (`false`).

Since there are five message types which can be sent as a request by the Content Manager, we give five use cases to make clear when which particular messages are being sent and how they are responded by the Scenario Manager.

Use case UC1: Selecting a menu item

Primary actor: Scenario Manager
Related actors: Content Manager, user
Brief description: A user would like to enter a particular menu or select a par-

ticular product and has selected the item at the user interface. The Scenario Manager verifies if the user is authorized to enter the menu or view the product. If so, the menu or product in question is returned. Otherwise, depending on the menu, the user has to profile or log in first.
Precondition: The Content Manager has sent a MenuChoice message to the Scenario Manager.
Success Guarantee: The Scenario Manager either returns the requested menu or product by sending a MenuRequest message or lets the user profile by sending a ProfileRequest message or lets the user login by sending a LoginRequest message.

Main Success Scenario:

- Scenario Manager receives a MenuChoice message from the Content Manager and reads the itemID and sessionID attributes.
- Using the sessionID attribute the user's current menu is determined. If the session is not found, a new one is created with default values.
- The itemID is stored as the *last menu choice*.
- Scenario Manager fetches menu data from the menu structure using the itemID attribute.
- Scenario Manager follows one of the following paths:
  - User has not logged in yet and logging in is either mandatory or advised.
    * Scenario Manager generates a LoginRequest object and sets the mandatory attribute to either `true` or `false`.
    * Scenario Manager constructs a LoginRequest message and sends it to the Content Manager.
  - Customer hasn't profiled yet and profiling is either mandatory or advised.
    * Scenario Manager generates a ProfileRequest object and sets the mandatory attribute to either `true` or `false`.
    * Each question is added to the profileQuestions attribute.
    * Scenario Manager constructs a ProfileRequest message and sends it to the Content Manager.

- Otherwise, the user is already authorized to enter the menu or view the product. In case of a menu:
  * Scenario Manager generates a MenuRequest object and sets the menuID attribute to the value of itemID (the requested menu).
  * Scenario Manager determines a list of products and priorities for the user and stores them in respectively the itemIDs and priorities attributes.
  * Scenario Manager constructs a MenuRequest message and sends it to the Content Manager.
- In case of a product:
  * Scenario Manager generates a MenuRequest object and sets the menuID attribute to the value of itemID (the requested product).
  * Scenario Manager determines a list of products and priorities for cross and up selling and stores them in respectively the itemIDs and priorities attributes.
  * Scenario Manager constructs a MenuRequest message and sends it to the Content Manager.

Use case UC2: Logging in a user

Primary actor: Scenario Manager
Related actors: Content Manager, user
Brief description: A user would like to login and has selected the login option at the user interface or the login screen is presented to the user because he would like to enter a particular menu. The authentication is performed by the Content Manager. When the user has successfully logged in at the Content Manager, the Scenario Manager has to be synchronized and has to return a personalized menu for the logged in user.
Precondition: The Content Manager has sent a `LoginAction` message to the Scenario Manager.
Success Guarantee: Scenario Manager has registered that the user is logged in and a personalized version of his current menu or the menu he would like to enter is sent to the Content Manager using a `MenuRequest` message.

Main Success Scenario:

- Scenario Manager receives a LoginAction message from the Content Manager and reads the customerID and sessionID attributes.

- If no profile is available for the user, a new one is created.
- Using the sessionID attribute the user's last menu choice is determined.
- Scenario Manager generates a MenuRequest object.
- Scenario Manager determines a list of products and priorities for the user and stores them in respectively the itemIDs and priorities attributes.
- The menuID attribute is set to the ID of the last menu choice.
- Scenario Manager constructs a MenuRequest message and sends it to the Content Manager.

Use case UC3: Logging out a user

Primary actor: Scenario Manager
Related actors: Content Manager, user
Brief description: A user would like to log out and has selected the logout option at the user interface. When the user has successfully logged out at the Content Manager, the Scenario Manager has to be synchronized and has to return a new, more general, menu for the logged out user.
Precondition: The Content Manager has sent a LogoutAction message to the Scenario Manager.
Success Guarantee: Scenario Manager has registered that the user is logged out and either a general version of his last menu or the main menu is sent to the Content Manager using a `MenuRequest` message.

Main Success Scenario:

- Scenario Manager receives LogoutAction message from the Content Manager and reads the sessionID attribute.
- Using the sessionID attribute the user's current menu is determined.
- Scenario Manager generates a MenuRequest object.
- Scenario Manager determines a list of products and priorities for the user and stores them in respectively the itemIDs and priorities attributes.
- The menuID attribute is set to the ID of user's current menu.
- Scenario Manager constructs a MenuRequest message and sends it to the Content Manager.

Use case UC4: Profiling a user

Primary actor: Scenario Manager
Related actors: Content Manager, user
Brief description: A user has to fill in a profiling form in order to receive advise about particular products. The form is shown to the user by the Content Manager. When the user has successfully filled in the form, it is sent to the Scenario Manager, which then returns a menu with products based on the answers provided by the user.
Precondition: The Content Manager has sent a ProfileAction message to the Scenario Manager.
Success Guarantee: Scenario Manager has registered that the user is profiled and a menu with products based on the answers provided is returned to the Content Manager using a `MenuRequest` message.

Main Success Scenario:

- Scenario Manager receives a ProfileAction message from the Content Manager and reads the sessionID and profileAnswer attributes.
- Using the sessionID attribute, the user's last menu choice is determined.
- The profileAnswers attribute contains the questions asked and the corresponding answers. If the user has logged in, the answers provided are stored in his profile.
- Scenario Manager generates a MenuRequest object.
- Scenario Manager determines a list of products and priorities for the user and stores them in respectively the itemIDs and priorities attributes.
- The menuID attribute is set to the ID of the last menu choice.
- Scenario Manager constructs a MenuRequest message and sends it to the Content Manager.

Use case UC5: Unprofiling a user

Primary actor: Scenario Manager
Related actors: Content Manager, user
Brief description: A user no longer want his profiling information to be used and has selected the unprofile option at the user interface. When the user has successfully unprofiled at the Content Manager, the Scenario Manager has to be

synchronized and has to return a new, general, menu for the unprofiled user .
Precondition: The Content Manager has sent a UnprofileAction message to the Scenario Manager.
Success Guarantee: Scenario Manager has registered that the user is unprofiled and either a general version of his last menu or the main menu is sent to the Content Manager using a `MenuRequest` message.

Main Success Scenario:

- Scenario Manager receives UnprofileAction message from the Content Manager and reads the sessionID attribute.
- Using the sessionID attribute the user's current menu is determined.
- Scenario Manager generates a MenuRequest object.
- ScenarioManager follows one of the following paths to determine the right menu for the user.
  - The user's current menu does not require the user to be profiled.
    - Scenario Manager generates a MenuRequest message, containing the current menu and standard products.
  - Otherwise.
    - Scenario Manager generates a MenuRequest message, containing the main menu and standard products.
- Scenario Manager sends the MenuRequest message to the Content Manager.

Composite use cases

Each of the five use cases above only deals with one request-response action. One message is received by the ScenarioManager and one is returned. It is possible that more of these interactions are required to perform some action. For example, when a user selects a menu and he has to log in before he can enter the menu. In such circumstances a combination of more use cases is required. This results in a composite use case. Combining use cases into a composite use case is straight forward. If the message that is returned by the Scenario Manager is of the MenuRequest type, a menu is returned and the use case is finished. If a message of the type LoginRequest or ProfileRequest is returned, the Content

Manager should answer with respectively a LoginAction or ProfileAction message. In case of the user mentioned in the example above, the composite use case looks as follows:

1. The user selects a menu, so use case *UC1: Selecting a menu item* is started.
2. Since the user needs to log in, a *LoginRequest* message is returned.
3. The Content Manager lets the user log in and sends a LoginAction message confirming that the user has logged in successfully.
    - If for whatever reason the logging in is not successful, no LoginAction message is returned and nothing happens.
4. Use case *UC2: Logging in a user* should be followed now, since it takes a LoginAction message as input.
5. A MenuRequest message is returned and the customer enters the menu.

3.3.2 Data Model

In order to realize the design, a data model is needed. The data model describes the logical data structure that can be used to create the databases. Figure 3.10 shows the data model used by the sales support system.

The components in the data model have the following meaning:

CUSTOMER Stores personal data about a customer

PRODUCT_OWNED Product that is owned by a customer

PRODUCT_TYPE Products which together form the portfolio of the company

MENU Stores information about menu screens

SESSION Stores session information

PROFILE_TYPE Stores various types of profile forms

PROFILE Stores profile forms

SEGMENT Stores segments for grouping customers.

PURCHASING_LIKELIHOOD Combines a segment, product and profile answer into an indication of how likely it is that a customer will buy a product.

AVAILABILITY Stores the availability of products in the zip code regions

ZIP_CODE Contains all the zip codes of the Netherlands.

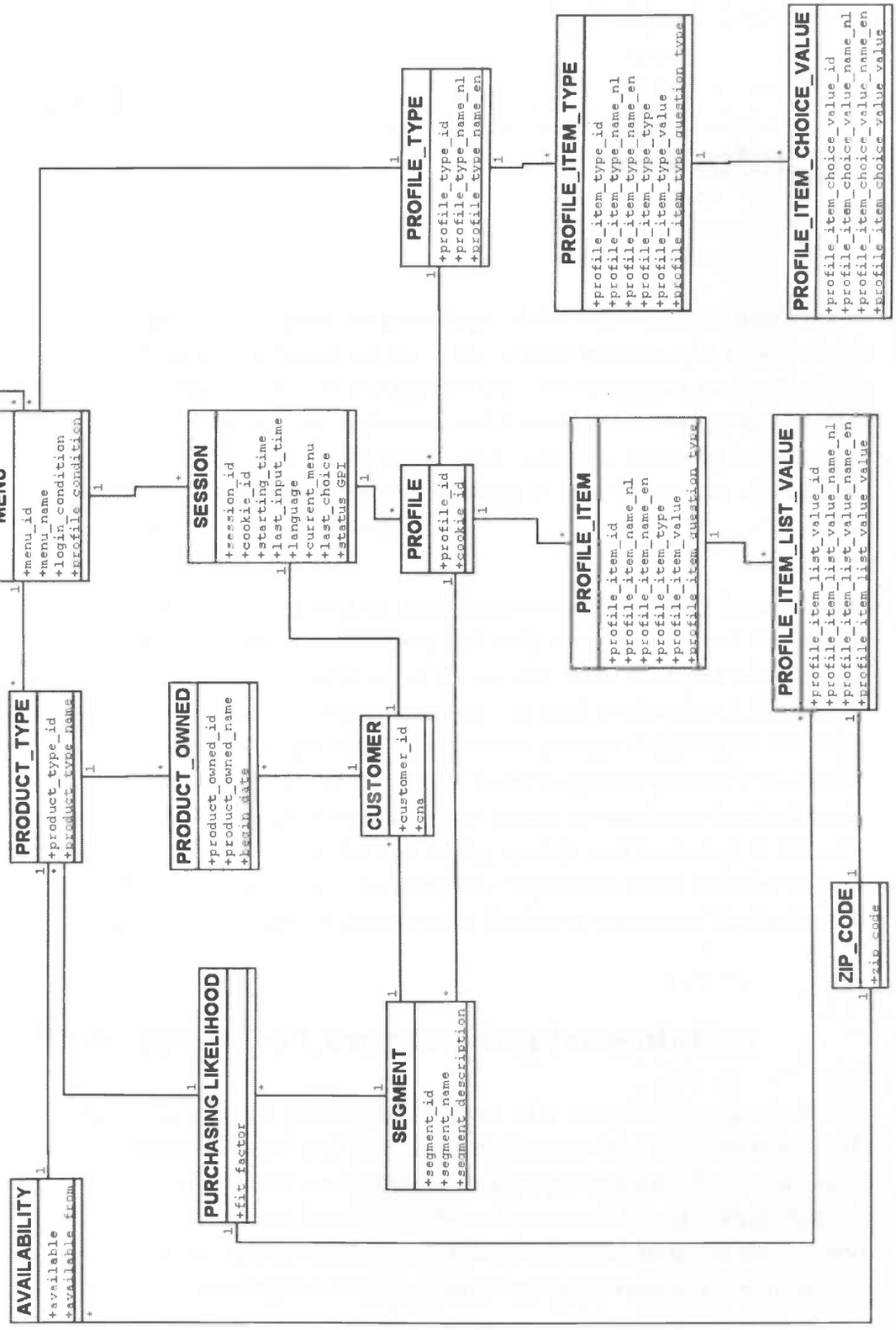

Figure 3.10: Data model

Chapter 4

Prototype

Using the design given in chapter 3 a prototype of the sales support system has been built. The prototype is based on the e-commerce website of a customer of Atos Origin Tum which is one of the largest telecom operators in the Netherlands. Using the prototype both technical and business concepts were tested. While most of the website's content (menus and products) has been included in the prototype, only a small selection will be used in this chapter as an illustration of the concepts introduced in chapter 3.

The prototype has been created in three iterations. After each iteration the results were reviewed. These results were not only used to improve the prototype itself, but also the requirements and the design were changed when ideas tried out using the prototype were successful. As said in chapter 3, the front-end of the prototype has been implemented by two groups of students. The first iteration of the prototype only contained the sales supports system's back-end with a very limited front-end. When the first group of students finished their front-end, it was integrated with the rest of the system and included in the second iteration. The third iteration contained the improved front-end developed by the second group of students together with the latest version of the back-end.

4.1 Prototype vs. a Complete Implementation

The advantage of making a prototype in stead of a complete implementation is its short development time and low development costs. This has been realized by laying the emphasize on the functional requirements. Note that most requirements in this thesis are functional. Non-functional requirements that are important in a eventual system and are not taken into account in the design and in this prototype include response time, reliability, robustness and security.

4.2 Technical Specifications

The prototype has been implemented in Java, using the J2EE platform. Furthermore, the Oracle JDBC API was used for database access and the Apache Axis API for implementing SOAP. Both the frond-end and the back-end were deployed on a Apache Tomcat web server. An Oracle database server, which is part of the network infrastructure of Atos Origin TUM was used to store the database components mentioned in section 3.3.

The communication between the Content Manager and the Scenario Manager is realized using Web services technology. The Scenario Manager component is implemented as a set of Web services, which can be requested by the Content Manager component using RPC. The XML messages are transferred over HTTP using the SOAP protocol.

As said above, the request messages which are received by the Scenario Manager component are sent using RPC. In order to be able to send these messages, the Content Manager component must be able to invoke procedures of the Scenario Manager component. We have defined five procedures with the same name as the messages, except that they start with a lower case character. In this way, the remote procedures provided by the Scenario Manager component are called `menuChoice`, `loginAction`, `logoutAction`, `profileAction` and `unProfile`. All the messages are classes containing attributes which form the content of the messages.

In order to be able to send the message objects, they need to be serialized first. After being serialized, the message objects are sent as a parameter with the remote procedure call. The return type of the remote procedures is a general object type, or super class, which is typecasted later to a specific message type.

4.3 Results

The prototype was tested with both front-end and back-end running on one machine, on two machines in the same building and on two machines that were at a distance of several kilometers from each other. With the complete prototype running on the same computer the performance was good, meaning that the response time of the user interface was fast enough for real interactive experience.

As stated in [29], the performance of messaging over the Internet infrastructure decreases when two machines are further away from each other. We experienced this ourselves when testing on two different machines. The delay when browsing through the menu structure was about a second which is too much for real interactive experience. On two machines in a distance of several kilometers there was even more delay: up to five seconds. For the prototype this is not a very big problem, because the emphasis is on the functional requirements, but for a complete implementation it is not acceptable. This may become a problem since an eventual system is also expected to use messaging. And we think it is the messaging aspect that causes most of the delay.

Chapter 5

Conclusion

After analyzing various solutions for a loosely coupled architecture supporting viewing and manipulation of stateful elements, an architectural concept based on Message-based MVC seemed to be the best solution. However, for practical reasons, the architecture that was eventually created is not based on M-MVC, but on RPC.

The research question formulated at the beginning of this thesis was:

Which software architecture is suitable for the design of a self-learning system that provides personalized business-customer interaction on various devices over the Internet infrastructure considering time-to-market, development costs, flexibility, maintainability and reusability?

We agreed that loose coupling of a software architecture increases the quality attributes mentioned in the research question. In our architecture described in chapter 3 we have included the Learning Machine component for the self-learning aspect of the system. The various device specific interfaces in the architecture make it possible to use a wide range of devices. The personalized business-customer interaction is realized by the Micro Funnel concept. In the architecture, the Micro Funnel is supported by the Scenario Manager component.

Using the design, a prototype based on the e-commerce website of a large telecom operator was created. Using this prototype the requirements of the sales support system as well as its performance were tested. The final version of the prototype has been used to demonstrate the functionality of the sales support system to the staff of Atos Origin TUM.

5.1 Results

As part of a presentation about the sales support system at Atos Origin TUM, the prototype has been used to demonstrate the functionality to the staff . The people involved were enthusiastic about the Micro Funnel concept. Atos Origin TUM would like to use the prototype to advertise in order to find one or more financiers. If one or more financiers have been found, the sales support system may be realized. It then most likely will become an extension of the existing OrderManager Suite.

Compared to the Message-based MVC approach, the coupling of model and view in our architecture of the sales support system is still too tight. Furthermore it is not event-driven. Both M-MVC and the architecture designed in this thesis use Web services, but in a different way. M-MVC uses the message-based SOA approach, while our architecture uses the RPC approach. M-MVC uses message oriented middleware. This middleware is event-driven and asynchronous. We have constructed our own messages for communication between the Content Manager and Scenario Manager components, but we use RPC as middleware for sending them. This is a request/ response mechanism which is synchronous. This is the reason why our architecture still too tightly coupled, compared to the M-MVC approach.

5.2 Recommendations

A drawback of the current version of the Micro Funnel is that the profiling form which has been used for the prototype is fixed. This implies that it can only be used to support particular scenarios. When the sales support system is implemented completely, forms should be dynamic and adapted to the context. Note that this functionality is also not present in the design. Nevertheless, this form is a good basis for testing how the menus become more personalized while the user moves through the Micro Funnel.

In order to realize a looser coupling between the Content Manager and Scenario Manager components, we recommend to replace the RPC middleware. Instead, a solution that uses the observer pattern [19] can be used. This asynchronous pattern supports loose coupling between components. In fact, both traditional MVC and M-MVC approaches use the observer pattern. It includes a

subject that is observed by a number of observers. The Scenario Manager would then become the subject, which is observed by the Content Manager. M-MVC uses an existing message-oriented middleware framework, called *NaradaBrokering* [27], to support a loose coupling between model and view.

If the sales support system is going to be implemented, we recommend to look at the possibility of replacing our current messages and their communication by RPC, by a message-oriented middleware framework like NaradaBrokering. We expect that, besides a looser coupling, the messaging performance will also increase.

Definitions and Abbreviations

Application programming interface (API) Interface that a computer system, library or application provides in order to allow requests for services to be made of it by other computer programs, and/or to allow data to be exchanged between them.

CRM (customer relationship management) Information industry term for methodologies, software, and usually Internet capabilities that help an enterprise manage customer relationships in an organized way.

Cross-sell Marketing term for the practice of suggesting related products or services to a customer who is considering buying something.

Customer segmentation Practice of dividing a customer base into groups of individuals that are similar in specific ways relevant to marketing, such as age, gender, interests, spending habits, and so on.

Explicit message Message where the data field carries both protocol information and instructions to be performed. Nodes must interpret each message, execute the requested task, and generate responses. These messages are highly flexible, varying in both size and frequency.

Implicit message Message that is less flexible than explicit messages, but have low overhead. The meaning of the data is defined in advance or selected at the time the connection is established. Therefore, processing time in the node is minimized during run time.

Interoperability Capability of two or more components or component implementations to interact.

Java Remote Method Invocation (Java RMI) Java API for performing the object equivalent of remote procedure calls.

Message-oriented middleware (MOM) Inter-application communication software that generally relies on asynchronous message-passing as opposed to a request/response metaphor.

Middleware Computer software that connects software components or applications. It is used most often to support complex, distributed applications.

One-to-one marketing CRM strategy emphasizing personalized interactions with customers.

Protocol Convention or standard that controls or enables the connection, communication and data transfer between two computing endpoints.

Remote procedure call (RPC) Protocol that allows a computer program running on one computer to cause a subroutine on another computer to be executed without the programmer explicitly coding the details for this interaction.

SOAP Protocol for exchanging XML-based messages over a computer network, normally using HTTP.

Up-sell Marketing term for the practice of suggesting higher priced products or services to a customer who is considering a purchase.

Bibliography

[1] H.W. Lie and J. Saarela, *Multipurpose Web publishing using HTML, XML, and CSS*, ACM, vol 42, nr 10, 1999

[2] Institute of Electrical and Electronics Engineers, *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*, New York, 1990

[3] J.C. Wortmann, *Interview given at the University of Groningen*, April 20, 2006

[4] M. Friedewald and O. Da Costa, *Science and Technology Roadmapping: Ambient Intelligence in Everyday Life (AmI@Life)*, European Science and Technology Observatory (ESTO), 2003

[5] D. Wu, I. Im, M. Tremaine, K. Instone and M. Turoff, *A Framework for Classifying Personalization Scheme Used on e-Commerce Websites*, Proceedings of the 36th Hawaii International Conference on System Sciences, 2003

[6] J.B. Schafer, J. Konstan and J. Riedl, *Recommender Systems in E-Commerce*, Proceedings of the 1st ACM conference on Electronic Commerce, ACM Press New York, 1999

[7] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice*, 2nd edition, Addison-Wesley, 2003

[8] R. Hilliard, *Views and Viewpoints in Software Systems Architecture*, Position paper from the First Working IFIP Conference on Software Architecture, San Antonio, 1999

[9] D. Garlan and M. Shaw, *An Introduction to Software Architecture*, Advances in Software Engineering and Knowledge Engineering, vol 1, World Scientific Publishing Company, New Jersey, 1993

[10] D.D. Corkill, *Blackboard Systems*, AI Expert, vol 6, nr 9, pp 40-47, September 1991

[11] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation, University of California, 2000

[12] *Thin Client*, article from Wikipedia, the free encyclopedia, 2006
`http://en.wikipedia.org/wiki/Thin_client`

[13] A. Kobsa, *Generic User Modeling Systems*, Journal of User Modeling and User-Adapted Interaction, vol 11, nr 1, Springer, 2001

[14] 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

[15] R. Pawson and R. Matthews, *Naked Objects*, ISBN 0470844205, John Wiley & Sons, 2002
`http://www.nakedobjects.org/book/content.html`

[16] T. Reenskaug, *Thing-model-view-editor: an example from a planningsystem*, technical note, Xerox PARC, May 1979

[17] T. Reenskaug, *Models - Views - Controllers*, technical note, Xerox PARC, December 1979

[18] *Model-View-Controller*, article from Wikipedia, the free encyclopedia, 2006
`http://en.wikipedia.org/wiki/Model-view-controller`

[19] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, ISBN 0-201-63361-2, Addison-Wesley, Boston, USA, 1995

[20] Java BluePrints, *Model-View-Controller*, Sun Microsystems, 2002
`http://java.sun.com/blueprints/patterns/MVC-detailed.html`

[21] M. Papazoglou, *Service-oriented computing: concepts, characteristics and directions*, Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03), pp 3-12, 2003

[22] *Enterprise service bus*, article from Wikipedia, the free encyclopedia, 2006
`http://en.wikipedia.org/wiki/Enterprise_Service_Bus`

[23] *Service-oriented architecture*, article from Wikipedia, the free encyclopedia, 2006
`http://en.wikipedia.org/wiki/Service-oriented_architecture`

[24] W3C, *Web Services Architecture*, W3C Working Group Note, February 11, 2004
`http://www.w3.org/TR/ws-arch/`

[25] *Web service*, article from Wikipedia, the free encyclopedia, 2006
`http://en.wikipedia.org/wiki/Web_service`

[26] *Publish/subscribe*, article from Wikipedia, the free encyclopedia, 2006
`http://en.wikipedia.org/wiki/Publish/subscribe`

[27] S. Pallickara and G.C. Fox, *NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*, Proceedings of the 4th International Conference on Middleware (Middleware'03), LNCS, vol 2672, pp 41-61, Rio de Janeiro, Brazil, June 2003

[28] *Java RMI Specification*, Sun Microsystems, 2006
`http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiTOC.html`

[29] X. Qiu, *Building Desktop Application with Web Services in a Message-based MVC Paradigm*, Proceedings of the IEEE International Conference on Web Services (ICWS'04), pp 765-768, 2004

[30] X. Qiu, *Message-based MVC Architecture for Distributed and Desktop Applications*, Dissertation, EECS Department, Syracuse University, 2005

[31] R.B. Miller, S.E. Heiman and T. Tuleja, *Strategic Selling: Secrets of the Complex Sale*, ISBN 1-85091-675-6, 1-85091-951-8 pbk, Kogan Page, London, 1988

[32] Business intelligence software toolkit from Business Objects, see `http://www.businessobjects.com`