A UML based hypermdeia system development methodology

Dollevoet, Stephan

Award date:
2002

TECHNISCHE UNIVERSITEIT EINDHOVEN

Department of Mathematics and Computer Science

MASTER'S THESIS

A UML Based Hypermedia System Development Methodology.

By

Stephan Dollevoet

Supervisors:	dr. M. Voorhoeve
	drs. L.H. verbeek
	ir. E.M.J.A. de Rooij

Eindhoven, Augustus 2002

ABSTRACT

Hypermedia applications are specific and mostly complex software systems. The navigational and presentational aspects of these applications require a different approach towards the development process than the development of traditional applications. The specific needs cause a shift of importance in the traditional development phases.

In this paper a methodology is presented for hypermedia system development based on the Unified Modelling Language (UML). UML is chosen because it is a widely accepted standard in modelling software applications. The presented methodology tries to capture the full life cycle of a hypermedia application by defining the workflow of the development process in terms of artefacts, tasks and actors.

The project was carried out as a research project by the order of 1eEurope NL to structure their development process. 1eEurope is a pan-European concern that develops e-business solutions for multinationals and implements Internet-related technologies for government agencies and non-profit organisations.

INDEX

1 INTRODUCTION

Hypermedia applications are specific and mostly complex software systems. The development of hypermedia applications differs from traditional software system development in that many people with different skills, such as multimedia designers, programmers and authors, are involved. Another main difference in their development is the increased role of the users of hypermedia applications, which adds more requirements to applications. Furthermore hypermedia systems tend to evolve from small into large applications that are very hard to maintain. Maintenance is a very important part in the life cycle of hypermedia development. However, little support is provided for structuring and modelling the specific properties of hypermedia systems.

There are some methodologies that cover part of the life cycle of the hypermedia development process. Some are based on entity-relationships models, such as HDM [GARZ93] and RMM [ISAK95]; others are based on object oriented models, such as OOHDM [SCHA98] and SOHDM [LEE98]. More recent proposals are based on the Unified Process [JACO99] and extend UML [OMG99]. Where Conallen [CONA99] focuses on the architecture, Baumeister, Koch and Mandel [BAUM99] propose a more model-based approach. Koch [KOCH00] presents a methodology that covers the full life-cycle of hypermedia systems development.

In this paper a methodology for the hypermedia systems development is proposed, based on the UML extensions proposed by Baumeister et al. [BAUM99]. UML is chosen because it is a widely accepted standard in modelling software applications. The extension of Baumeister et al. is chosen because it covers the conceptual, navigational and presentational aspects of the hypermedia application. Furthermore the roles to be played and tasks to be performed in the development process will be defined as well as the deliverables of each step in the development process.

Chapter 2 will discuss the life cycle of a hypermedia application and the difficulties that are encountered when developing such an application. In chapter 3 an extension of UML is presented, which allow hypermedia designers to model specific properties of hypermedia applications. The extension is illustrated with an example of a web based book shop. Using this extension requires a somewhat different approach towards designing and developing a hypermedia application. In chapter 4 a methodology is proposed, which offers a systematic approach towards hypermedia application development. Chapter 5 will present some conclusions and remarks towards the use of this methodology.

As an example we show the design process of a web shop. The shop lets you browse through a product catalogue. The user can also search the catalogue by title and author. On browsing the users can put products in the Shopping Cart. When the user has finished shopping, he can order all products that he has put in the Shopping Cart.

2 LIFE CYCLE OF (HYPERMEDIA) SYSTEM DEVELOPMENT

2.1 INTRODUCTION

The life cycle of hypermedia system development differs from the traditional software development life cycle by a shift in the importance within the different phases. For instance, in hypermedia system development the user interface plays a more important role. And as hypermedia systems evolve from small applications into large applications, maintenance plays a far more important role in the life cycle. In this chapter the life cycle of a hypermedia system is discussed, pointing out the differences with traditional software systems.

The life cycle of a software system can be roughly divided into five main phases:

- Analysis
- Design
- Realisation
- Introduction
- Maintenance

In the *analysis phase* a specification is made of the demands, wishes and expectations the customer/user has towards the application. Furthermore an evaluation is made of the environment in which the application has to perform its tasks and an evaluation of currently used applications or work processes, which have to be replaced or improved by the hypermedia application.

In the *design phase* the application is designed and modelled. The first step in the design phase is make a description of the behaviour of the application and its environment. This description is a transitional phase from the analysis towards a technical design. The technical design is a formal way to model the application to be implemented. Together with a detailed time schedule the technical design forms a blueprint for the realisation phase.

The *realisation phase* is the phase in which the application is physically build. The models created in the design phase are converted into code. After implementing the models, the generated code is subdued to a number of tests, which are performed to check whether the implementation satisfies the requirements captured in the analysis phase.

During the *introduction phase* the application is introduced in the environment it has to run and is tested by a group of end users after receiving training. In this phase it is verified whether the application matches the requirements and expectations of the end users. The tests performed by this group of end users often result in additional requirements, which also have to be implemented. After successful results, the application is accepted and will be released, and the development process now enters the maintenance phase.

In the *maintenance phase* adjustments are made to the application and additional functionalities are added. Every time additional functionalities are added, the four previous phases are executed again. So the maintenance phase can be viewed as consisting of a succession of system developments. As stated earlier, hypermedia systems often evolve from small into large applications and therefore require a lot of maintenance. Not only more functionality is added, content and lay out changes occur constantly, requiring different functionalities as well.

2.2 Analysis

The first phase in the life cycle is the analysis phase. The main goals of the analysis phase are to capture the requirements towards the application and to get a clear vision of the desired result. Based on this analysis, the risks can be assessed and a rough planning of the project can be made, so a decision can be made whether to continue with the project or not.

In this phase it is important to gather the necessary information. As the representation of information plays a more important role in hypermedia development than in traditional development, a thorough understanding is needed of the following issues:

- what kind of information has to be shown
- in what way should this information be presented
- in what way changes this information in time
- what is the relationship with other pieces of information

Often a hypermedia application replaces another (traditional) application or some working process. By carefully analysing this application or process, an understanding of the application to be built can be obtained. The properties, oddities and problems from the original application or process have to carefully studied, as the hypermedia application has to replace or improve the functionalities from the original application.

Other important factors are the people and resources involved. Hypermedia design differs from traditional design in the amount and diversity (of categories) of people that are involved in the development, like end users, software architects, software designers, software developers, graphical designers, project managers, marketing and sales people, providers etc. They all need and provide resources like information, expertise, tools and hardware. A clear vision of why, when and how long they are involved, is the key to a sound project planning which is one of the main deliverables of the analysis phase together with a risk and cost analysis.

All this information is gathered and analysed with one goal: to be able to make a decision whether to continue with the project. The go/no-go decision is in fact the milestone for the analysis phase. This milestone is preceded by verification. Verification is the process of checking and analysing the information gathered. When verification is successful, a decision will be taken to continue or not, which could be called validation. Verification and validation is done at the end of each step in the process. A successful verification leads to validation and marks the finishing point of a step in the process.

2.3 Design

The second phase of the life cycle is the design. This phase is divided into three sub-phases: functional design, technical design and graphical design. The functional design describes the behaviour of the application in an abstract way. In the technical design the classes and objects of the problem domain are defined together with classes and objects for technical infrastructure, which results in a detailed specification of the construction phase. The graphical design models the presentational aspect of the application.

2.3.1 Functional Design

In the requirements specification made during analysis, the functionalities are described in terms of concepts. A functional design defines these concepts in a detailed manner, describing the behaviour of the application in an abstract way. Koch [KOCH00] distinguishes two sorts of requirements: functional requirements and non-functional requirements.

The functional requirements describe the behaviour of the application, which can be classified as:

- Related to the content, e.g. evaluation of user input.
- Related to the structure, e.g. navigation through the content.
- Related to the presentation, e.g. presentation of the content.
- Related to the user profile, e.g. user preferences.

The non-functional requirements describe the properties of the hypermedia application, e.g. environment constraints, performance, extensibility, etc. Together the functional and non-functional requirements define the boundaries in which the application has to perform its tasks, defining the problem domain of the application.

2.3.2 Technical design

The technical design can be divided into two subphases. In the first subphase the classes and objects of the problem domain are defined. The classes and objects are limited to those that represent concepts recognizable to the customer, excluding those needed for communication, databases, interface, etc. In the second phase the results of the first phase are transformed into a technical solution. New classes and objects are to be designed for technical infrastructure. The classes and objects of the first phase are embedded into this infrastructure. This design results in a detailed specification of the construction phase.

A big difference with traditional design is the important role of the presentation of information in hypermedia design. A hypermedia application designer not only should have technical knowledge, but should also be aware of cognitive issues. And not only the way information is shown is important, but also the ways that information can be accessed. Users want easy access to the information they need and do not want to get lost when browsing hypermedia applications, which will only cause a decrease of interest and an increase of frustration.

2.3.3 Presentational design

Presentational design plays a far more important role in hypermedia design than it traditional design. Where in traditional design lay out was a minor issue of the application, in hypermedia design lay out is a major issue of the application. Hypermedia applications often have to satisfy corporate lay-out regulations and rules for the presentation in order to guarantee a uniform corporate identity. Multimedia objects like images, video and flash animations put a lot of requirements on lay out and performance.

2.4 Realisation

The realisation phase is the phase in which the application is physically built. This phase consists of two subphases: implementation and testing. In the implementation phase the design models are translated into code. In order to keep control over the implementation phase, the design is divided into units that can be separately developed and tested. After successful testing of these units, they are assembled to larger units, eventually arriving at a complete application.

On completing the application, it is subjected to a number of tests to check whether it satisfies the requirements defined in the analysis and functional design. A number of tests are distinguished:

- unit tests
- integration tests
- functional tests

Unit tests regard the classes/objects of the application. Integration tests check the integration of these units into a correct working subsystem. Functional tests are performed to check whether the system meets the required functionalities in the development environment.

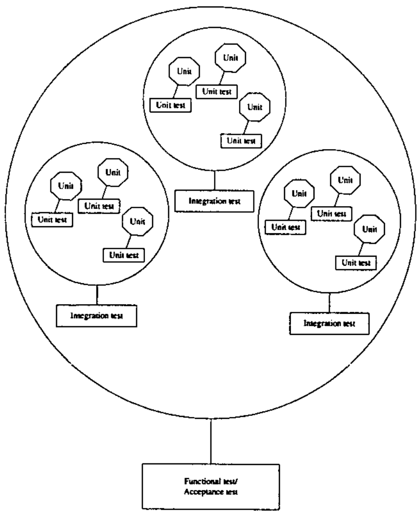


Figure 2-1 Verification & Testing

2.5 Introduction

During the *introduction phase* the application is introduced in the environment it has to run. A small group of end users verifies whether the application matches the requirements and expectations of the end users. Errors that occur are fixed and adjustments are made because functionalities are not satisfying certain requirements and expectations. This is repeated until the users provide positive results of the acceptance tests. Furthermore the end users receive training in handling the application. At the end of the introduction phase, the application is released. All the software, documents, models and user manuals are now completed. Often the application is evaluated to determine necessary adaptations during maintenance.

2.6 Maintenance

After the application is released, the maintenance phase starts. Additional functionalities are developed and the application is adapted to changes in the environment and resources. Whenever the additions or adjustments are considerable, the four previous phases are visited again, adding a small life cycle within the maintenance phase.

Hypermedia systems are highly evolutionary and therefore require a lot of maintenance. Hypermedia links cause the appearance of dangling and incorrect links, which must be avoided. The content and lay out of a hypermedia application usually change constantly, imposing different requirements on the application. Maintaining a hypermedia application is therefore a time-consuming process.

At some point in time, the life of the application ends. New technologies and changing work processes bring different requirements the application cannot fulfil anymore, so it is replaced.

3 UML EXTENSION FOR MODELLING HYPERMEDIA APPLICATIONS

3.1 INTRODUCTION

In this chapter a UML extension for modelling hypermedia applications will be shown. In this case UML is chosen for modelling hypermedia applications because it is a widely accepted standard throughout the world of software design. This paper is not a discussion whether UML is the best modelling language for modelling hypermedia applications.

The requirements a UML extension has to meet are discussed first. The design of UML opts for simplicity rather than completeness. This choice necessitates extensions to UML for specific uses.

Second, an extension, proposed in Koch [KOCH00], is discussed. This extension is intuitive and transparent and builds a model which can be easily implemented. The extension introduces a navigational and presentational model, which can be derived from a conceptual model, and will cover typical aspects of a hypermedia application.

3.2 EXTENDING THE UML

There are some requirements towards extensions of UML defined by the definition of UML [OMG99]. The definition provides three mechanisms to extend the UML: tagged values, constraints and stereotypes.

Tagged values are properties which can be assigned to the elements of the UML. Examples of tagged values are preconditions and postconditions assigned to operations in a class.

Constraints are rules that restrict the semantics of elements of the UML. Constraints are assigned to classes or objects, or are assigned to relations, constraining the classes or objects that participate in the relation.

Stereotypes are extensions of the semantics of existing modelling elements. When a stereotype is assigned to an object, its semantics is replaced by the semantics of the stereotype. The stereotype can be seen as an modelling element with added or new semantics.

Extending UML is only allowed by these three mechanisms, but there are some additional requirements towards extensions of UML. Since the UML is based upon an intuitive and easily understandable diagrammatic notation and this must hold for extensions as well. So the first requirement is that the diagrams, which are introduced by extensions, are easily drawn and comprehended.

A second requirement towards extensions is that they can be easily mapped upon other diagrams. Since UML diagrams are all different views of the same system, the extension is yet another view on the same system. All these views must be consistent, and together cover all aspects that need modelling.

The third requirement is the implementation of the extension into a modelling tool is easy. If the extension would be difficult to implement in a modelling tool, it would not only take a lot of effort just to be able to use the extension for modelling a system, but switching from one tool to another would be impossible.

The last, but certainly not least requirement towards an extension is that it is transparent. The semantics has to be crystal clear. If not, the extension would only bring more confusion into the UML model than structure.

Several UML extensions for hypermedia development meeting these requirements are available. Conallen [CONA99] presents an extension for Web modelling. This extension is a set of stereotypes for typical hypermedia objects. However, the extension is more suited for implementation aspects than it is for designing aspects. Another extension is proposed by Koch [KOCH00]. This is a more intuitive approach and focuses more on the navigational and presentational aspects, which are typical for hypermedia applications.

3.3 AN EXTENSION FOR HYPERMEDIA APLLICATION

The extension proposed by Koch consists of three iterative design processes:

- Conceptual Modelling
- Navigational Modelling
- Presentational Modelling

1
The conceptual model is constructed on the traditional way of building an object-oriented model, by defining the classes and their relations and their constraints. It is based on functional requirements described and defined by use cases. Conceptual models are represented by class diagrams.

Navigational design focuses on the navigational aspects of hypermedia applications and consists of two models:

- A navigation space model
- A navigational structure model

The navigation space model can be derived from the conceptual model. The navigation space model is a class diagram as well, and it consists of classes which can be visited during navigation. From the navigation space model the navigational structure model is constructed. The navigational structure model shows how navigation can be done. For these purposes a specification is made for an automatic generation of the navigational model out of the conceptual model.

Presentational design focuses on the presentational aspects of the hypermedia application and consists of the following:

- Storyboard scenarios
- A static presentational model

- A dynamic presentational model

The presentational model is derived from the navigational structure model and the requirements imposed on the presentation. As the navigational structure shows how can be navigated towards classes, the presentational model shows what will be shown when navigated to these classes.

The requirements towards the presentation are represented by storyboard scenarios. Storyboards are sketches of the lay out of the user interface. Storyboard scenarios are possible sequences of storyboards, representing navigation through the application. From these storyboard scenarios a static presentational model is constructed, which model elements used for presentation. Besides a static presentational model a dynamic presentational model is constructed, which is a UML sequence diagram. The dynamic presentational model represents the flow of the presentation.

In this way a sound model can be created in an automated way, covering the specific properties of hypermedia applications. In the following sections these models will be discussed and explained by an example. Step by step, from the conceptual model through the navigational model towards the presentational model, the design will be created.

3.4 The Conceptual Design

The conceptual design in UML is created from use cases. Use cases are a modelling technique in which the requirements are captured. It is built in an iterative process between designers and the customers/end users and presents the functionalities of the application in an intuitive manner. Use case models consist of use cases, actors and their relationships.

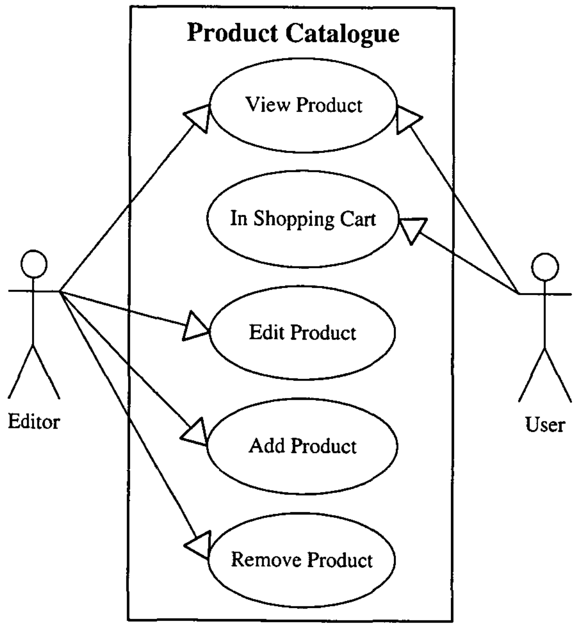


Figure 3-1 An Example Of A Use Case

An actor is someone or something that interacts with the system. The actors carry out the use cases. Use cases are a set of actions performed by the system that presents a result to some actor. Together with a clear definition of the boundaries of the system, the actors and use cases present a clear requirement specification.

The conceptual design builds a model based on the use cases. The conceptual model is a graphical representation of the classes of the system and the relations between them. It is a UML class model. It defines the classes and its attributes and operations and the associations between them. It also defines the constraints, dependencies and the hierarchical structure.

3.5 The Navigational Design

The next step is from the conceptual design to the navigational design. First the space in which can be navigated has to be modelled. A hypermedia application consists of nodes and links between these nodes. The links are in fact relationships of type 'navigation' and connect 'navigational' objects or classes. These navigational classes can be obtained from the classes from the conceptual class model.

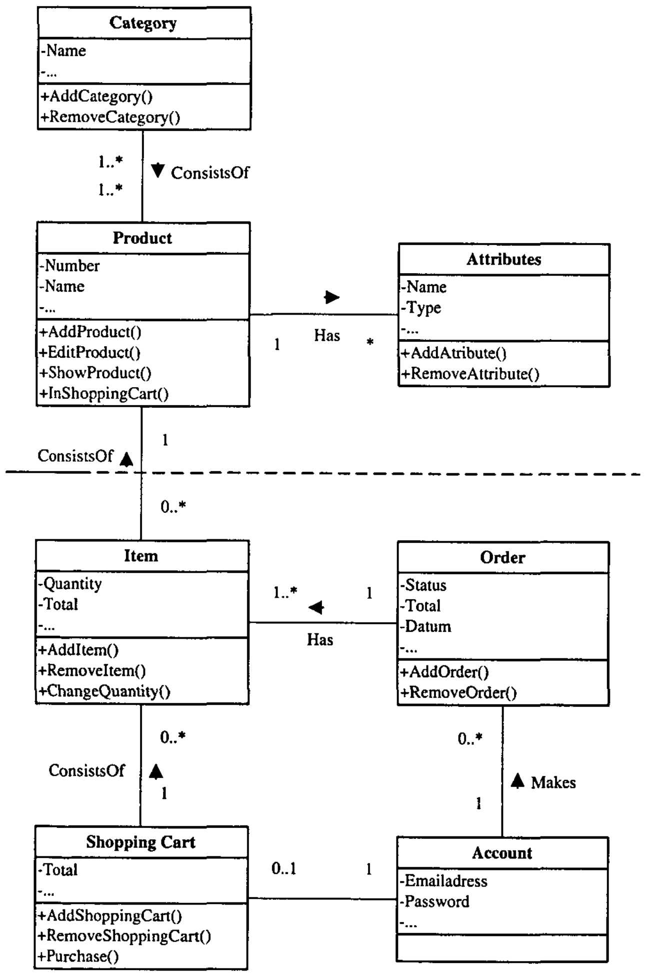


Figure 3-2 Conceptual Class Model

The second step is to model the navigational structure. The design of the navigational structure is an often neglected design step. Even small hypermedia applications that have few nodes often have a complex navigational structure and the complexity increases as the application grows. The more links there are, the more there can be navigated through the application. But more navigation certainly does not mean better navigation. Someone browsing through a complex hypermedia application with a lot of links can easily lose his or her way in the application.

So by modelling the navigation space and structure, a clear view of the way that can be navigated to certain information can be obtained. The navigation space is modelled by the navigation space model and the structure by the navigational structure model.

3.5.1 The Navigation Space Model

The navigation space model can be obtained from the conceptual class model, and is also a UML class model. The navigation space model defines the space in which can be navigated, thus it models the nodes and the links of the application. Every class of the conceptual model that can be visited through navigation is a node in the navigation space model. For the representation of these classes a UML stereotype ‹‹ *navigational class* ›› is introduced which is shown in Figure 3-3.

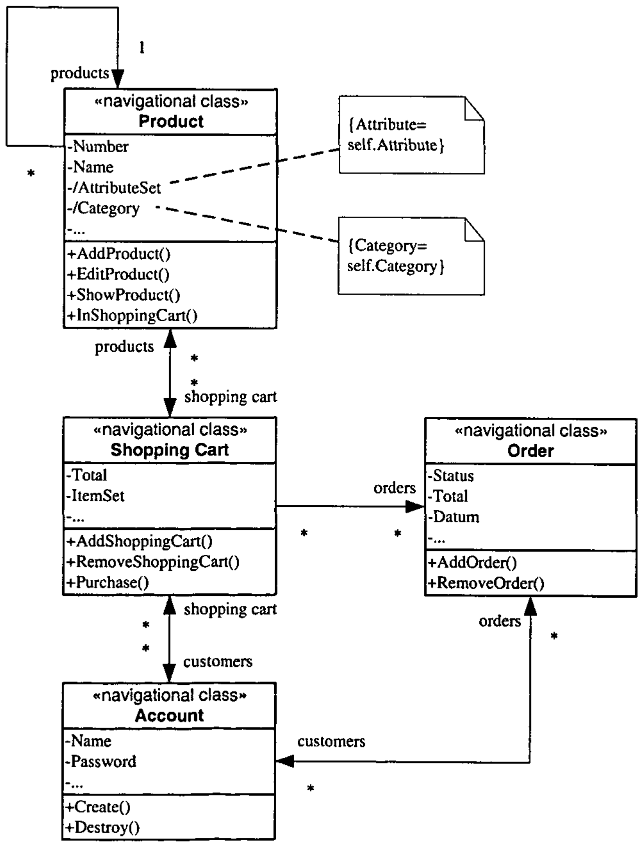


Figure 3-3 Navigation Space Model

Each class has the same name as its corresponding class in the conceptual model. Classes of the conceptual model not needed for navigation are omitted or are represented as attributes of other classes. The values of the attributes can be computed from the conceptual objects, which can be done by defining a constraint. The attributes are called *derived attributes* and can be represented in UML by a slash (/) before its name. In Figure 3-3 the class *Attribute* is reduced to a derived attribute of the class *Product.*

The associations between these navigational classes represent the links between the nodes in the hypermedia application. These associations represent *direct navigability* from one navigation class to another. Hence their semantics differ from the associations used in the conceptual model. To represent the direction of the navigation the associations are directed. This is shown with an arrow attached to the directed end, or maybe two ends if it is bi-directed. Each directed end is provided with a name and an explicit multiplicity. Associations of the conceptual model not needed for navigation are omitted.

Figure 3-3 shows the navigation space model for the conceptual model shown in Figure 3-2. As you can see the navigational space model is a subgraph of the conceptual class model. Every class in the navigation space model is also a class in the conceptual model.

3.5.2 THE NAVIGATIONAL STRUCTURE MODEL

The navigational space model is a graphical representation of the direct navigation from one object to another object in the problem domain. It contains no information about how navigation is done. The *navigational structure model* defines the navigation and it is based on the navigation space model.

Navigation between instances of the navigational classes, the navigational objects, can be performed in several different ways. , i.e. menus and indexes. To distinguish these possible ways, additional modelling elements are introduced: *menus*, *indexes*, *external nodes* and *navigational contexts*. The concept of *navigational nodes* is introduced to refer to these modelling elements and the navigational objects.

3.5.2.1 NAVIGATIONAL CONTEXT

The concept of *navigational context* was introduced in OOHDM by Schwabe and Rossi [SCHA96] and is defined as a sequence of navigational nodes. It defines links from each navigational node in a navigational context to a previous and next navigational node in the same navigational context. Additionally the first and last node or sometimes a circular navigation are defined. In this way different groupings of navigational objects are possible. For example, in our web shop the book "The count of Monte Christo" is shown as one of the "books by Alexandre Dumas" and as one of the "classical books".

A navigational context is represented by a UML stereotyped object ‹‹ navigational context ›› and an OCL expression defining the sequence of the navigational objects. There are three different kinds of navigational contexts: navigational (or simple) contexts, grouped contexts and filtered contexts. Simple contexts are contexts where all navigational objects of the class are grouped together and are ordered by an attribute, for example "all books" ordered by name.

Grouped contexts are navigational objects grouped by common values of an attribute or common objects related by an association. For example, "all books by Alexandre Dumas" or "all classical books". Grouped contexts are represented by the stereotyped object ‹‹ grouped context ››, see Figure 3-4.

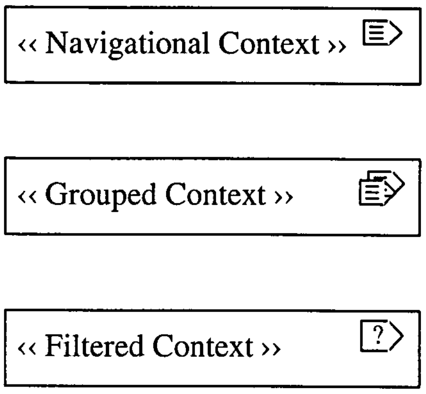


Figure 3-4 Navigational Context

Filtered contexts are navigational objects grouped in the same way as grouped contexts. The difference between the two is that the common values or objects are assigned dynamically, usually supplied by the user in a query that is part of the filtered context. For example, "all books containing the word count". Filtered contexts are represented by the stereotyped object ‹‹ filtered context ››, see Figure 3-4.

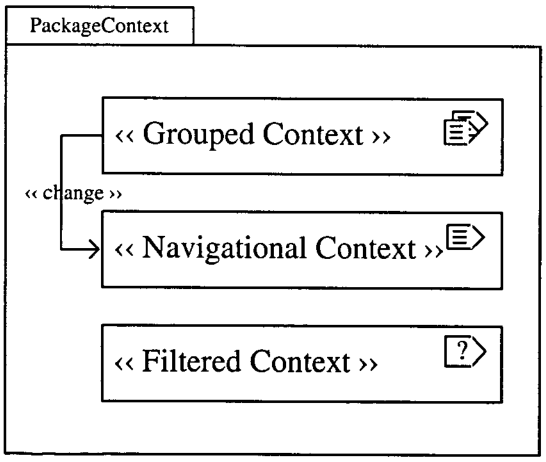


Figure 3-5 Package of Navigational Contexts

The navigational contexts of the same navigational class are grouped together in a UML package, see Figure 3-5. The contexts in a package are related by associations, which allow for context changes.

Context changes are possible for navigational objects that are part of both contexts. Context change between two navigational contexts is defined by the stereotyped association ‹‹ change ››, see Figure 3-5. For example, when visiting the book "The count of Monte Christo" in the context "classical books" one can change to the context of "books by Alexandre Dumas".

Figure 3-6 shows a navigational context for the navigational class Product.

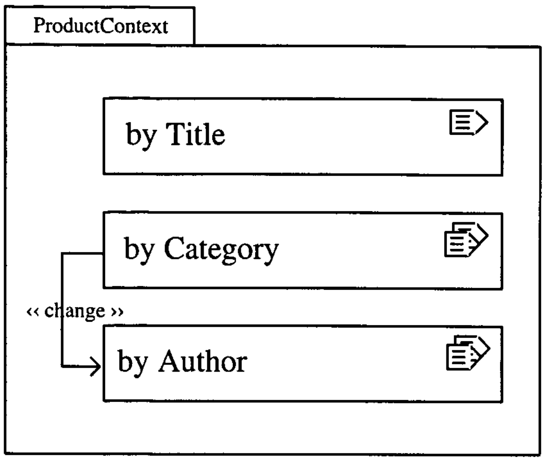
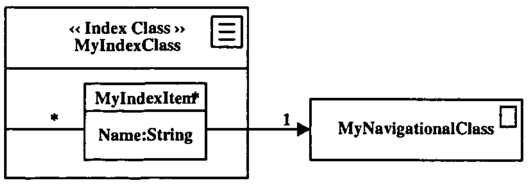


Figure 3-6 An Example Of Navigational Context for the class Product

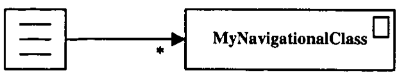
3.5.2.2 ACCESS ELEMENTS

Navigational context defined the way to navigate between the navigational objects. Access elements are defined to represent the way the navigational contexts are accessed. The access elements are *indexes*, *guided tours*, *queries*, *menus* and *external nodes*.

Indexes, guided tours and queries are all sets of objects of the same navigational context, from which the user selects one. Each object of such a set is an object with a name and a link to an instance of a navigational class or to an access element. An index is an unordered set of objects. An index is represented by a stereotyped class ‹‹ index ››, which is a composition of classes, see Figure 3-7a. Figure 3-7b introduces a shorthand notation for an index, where the association is derived from the two associations of the stereotyped class ‹‹ index ››.



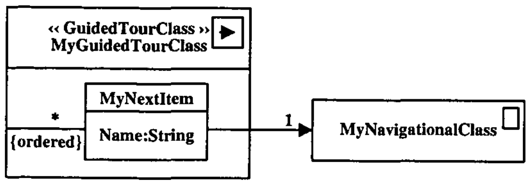
(a) Index Class



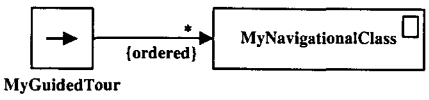
(b) Shorthand notation for Index Class

Figure 3-7 Index Class

A guided tour is an ordered set, providing a sequence in which the objects are navigated. The guided tour is represented by a stereotyped class ‹‹ guided tour ››, which is connected to a navigational class by a directed association with an *{ordered}* property, see Figure 3-8.



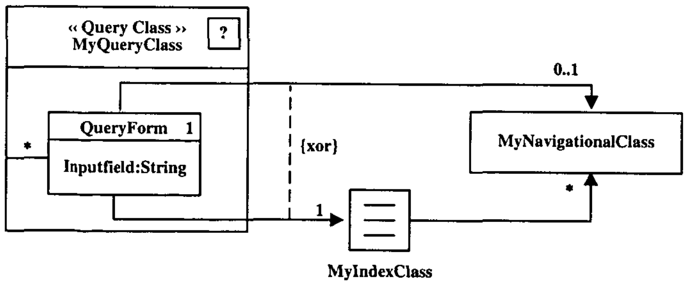
(a) Guided Tour



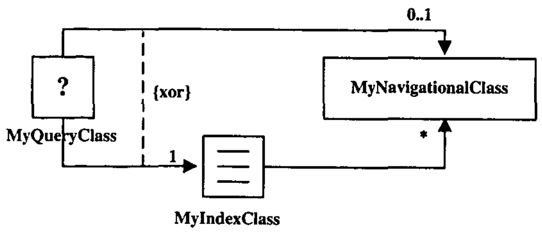
(b) Shorthand Notation for Guided Tour

Figure 3-8 Guided Tour Class

A query is a set of objects that are filtered by a query string. The query is represented by a stereotyped class ‹‹ query ››, with a query string as an attribute. When the query is evaluated it produces a filtered context. Often a query has multiple directed associations. When the evaluation of the query string leads to one result its navigational object is shown. When the evaluation of the query string leads to more than one result, the results are first shown in an index. After selecting one of the index items, the selected navigational object is shown. This is modelled by using a *{or}* constraint, so a distinction can be made in navigating to a navigation class depending on the number of results of the query string, see Figure 3-9.



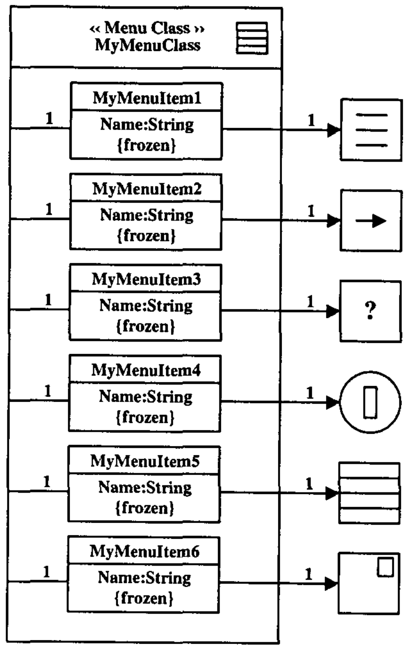
(a) Query



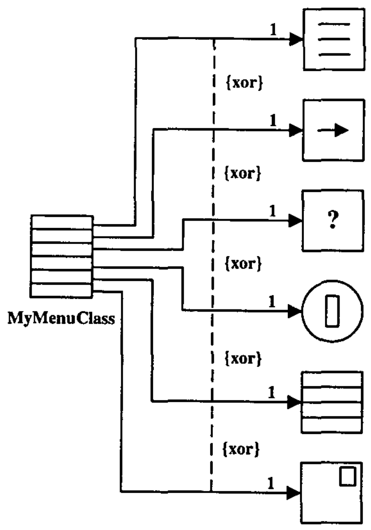
(b) Shorthand for Query Class

Figure 3-9 Query Class

A menu is a set of objects of different navigational context. These are the same objects as the objects of indices. The only difference is that the name of the object is a constant. A menu is represented by a stereotyped class ‹‹ menu ››, which is also a composition of classes, see Figure 3-10a. Figure 3-10b is shorthand for the stereotyped class ‹‹ menu ››.



(a) Menu Class



(b) Shorthand notation for Menu Class

Figure 3-10 Menu Class

An external node is a navigational node not belonging to the hypermedia application. It is represented by the symbol of Figure 3-11.

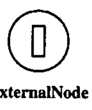


Figure 3-11 External Node

3.5.2.3 NAVIGATIONAL STRUCTURE MODEL

The navigational structure model can be derived from the navigation space model by modelling the navigational context for every navigational class in the navigation space model and connecting these navigational classes and contexts by modelling access elements between them. Figure 3-12 shows the result of this derivation from Figure 3-3.

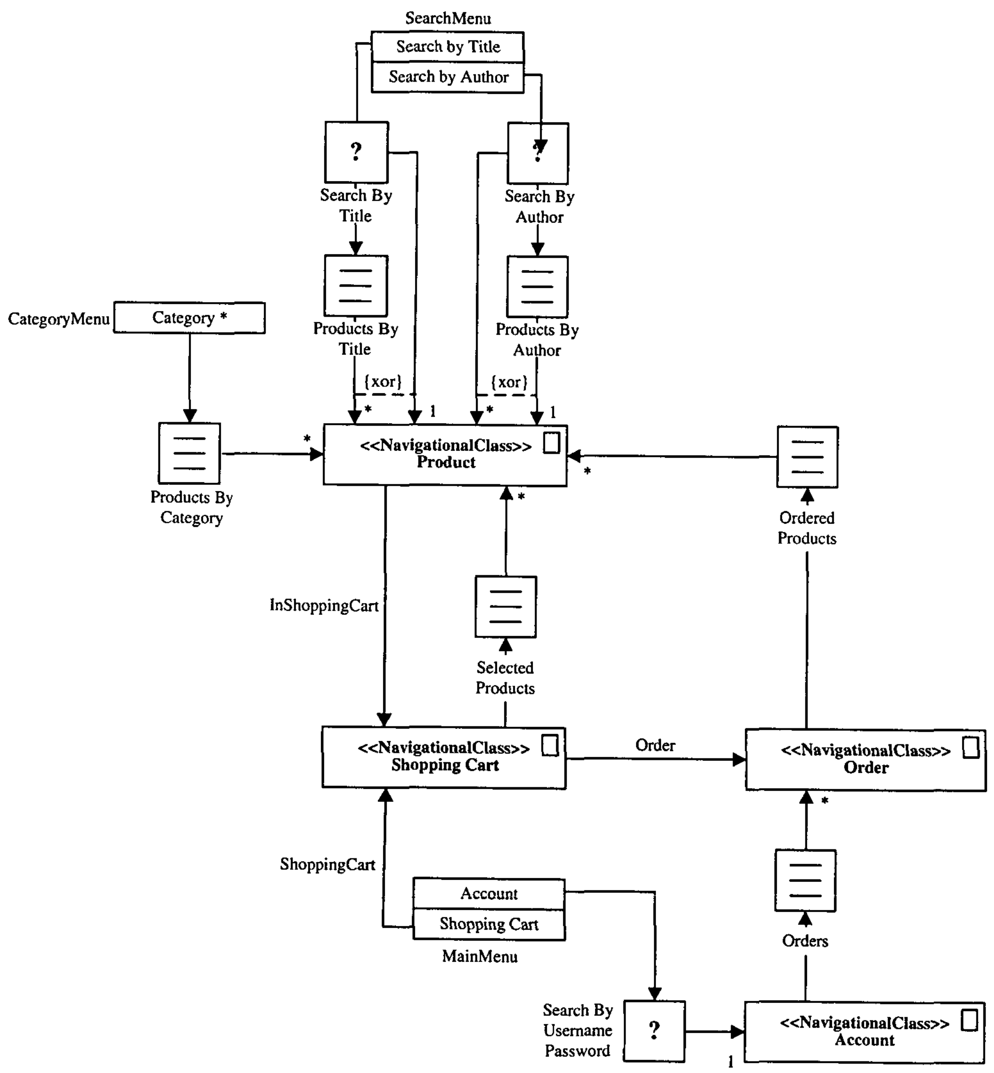


Figure 3-12 Navigational Structure Model

3.6 The Presentational Model

The construction of the navigational structure model does not model presentational aspects. It models how an object can be visited, but it contains no information about how this object is presented. All navigational nodes, both navigational classes and access elements, must be presented to the user.

There are two major presentational aspects: visual and structural. The visual aspects are lay-out characteristics such as colours, fonts, formats, etc. of the application. As visual aspects are very hard to model, they are usually designed by drawing sketches. This is usually done by a multimedia designer rather than a software developer. However rough sketches do not cover all the aspects of the presentation. To get a more clear vision of the user interface storyboarding is used. Using storyboarding helps to capture the requirements imposed on the user interface. Storyboarding is discussed in section 3.6.1.

The structural aspects of the presentation are elements like windows, frames, anchors, buttons, etc. These structural aspects focus more on the organisational aspects of the presentation. There are two different presentational models: a static model and a dynamic model. The static presentational model structures the presentational elements. It tells us what elements are presented in which view. The static presentational model is a UML class model and is discussed in section 3.6.2.

The dynamic presentational model models the behaviour of the presentational elements. It shows how the user interface changes when there was an interaction with the system, typically done by the user. The dynamic presentational model is modelled by a UML sequence diagram and is discussed in section 3.6.3.

3.6.1 Storyboarding

Requirements towards the visual aspects of a hypermedia application are often captured by the drawing of sketches. Though these sketches are very useful to get a notion of the look and feel of the application, they only contain a couple of the relevant elements of the presentation. To get a clear vision of the elements presented, storyboards are used. Storyboards are a useful help in defining the user interface, and are an addition to the sketches. Together they provide a clear vision of the user interface. Another advantage of using storyboards is that they can be used parallel to the functional and technical design.

3.6.1.1 User Interface Elements

The user interface of a hypermedia application consists of a number of user interface elements. User interface elements are UML stereotyped classes. They all have their own specializations modelling particular hypermedia interface elements. The user interface elements are modelled by the following stereotyped classes:

- *‹‹ text ››*
- *‹‹ anchor ››*
- *‹‹ button ››*

- *‹‹ multimedia ››*
- *‹‹ form ››*
- *‹‹ collection ››*
- *‹‹ anchored collection ››*
- *‹‹ external application ››*

The stereotype *‹‹ text ››* models a sequence of characters together with its formatting information.

The stereotype *‹‹ anchor ››* is a clickable user interface element with a link attached to it. It is the starting point of the navigational association described by the link. By clicking on it, one is navigated from one navigational node to another.

The stereotype *‹‹ button ››* is a clickable user interface element with an action or event attached to it. By clicking on the button, the action or event attached to it is performed. By using the button as an anchor, the button also can be used as a navigational starting point.

The stereotype *‹‹ multimedia ››* models multimedia objects like images, audio, video, flash animations, etc. They are often associated with other user interface elements to add some more functionality, such as start and stop buttons for video animations, or images with anchors to model clickable maps.

The stereotype *‹‹ form ››* is used for modelling user input. A form contains input fields, select boxes, checkboxes, etc. The stereotype *‹‹ form ››* defines the lay out of the form, the information presented, evaluation of the input and the triggers of the events.

The stereotypes *‹‹ collection ››* and *‹‹ anchored collection ››* are both composition classes used to group user interface classes. A *‹‹ collection ››* is a composition of a set of user interface elements. The stereotype *‹‹ anchored collection ››* is a composition of a set of anchors.

The stereotype *‹‹ external application ››* is used to bring user interface elements of other applications into the model. Typical external applications are ActiveX components and Java applets.

These are the most common user interface elements. Additional user interface elements are easily defined by introducing a new stereotyped class in the same way as the above, or by composition, or by subclassing on of the above. The user interface elements are presented in Figure 3-14.

3.6.1.2 THE PRESENTATIONAL CLASS

The presentational class is a UML composite object. The presentational class is a composition of the user interface elements of the previous section and is represented by a UML stereotyped class *‹‹ presentational class ››*. An example of the *‹‹ presentational class ››* is presented in Figure 3-13 and its semantics in Figure 3-14.

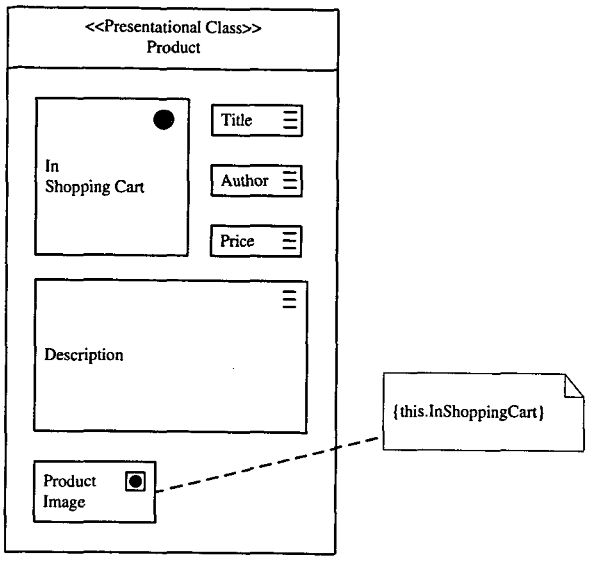


Figure 3-13 Example Of A Presentational Class

3.6.1.3 USER INTERFACE VIEW

The user interface view is also a UML composite object. The user interface view is a composition of presentational classes and represented by a UML stereotyped class *‹‹ user interface view ››*. The *‹‹ user interface view ››* is the window (and its content) presented to the user. It consists of a number of *‹‹ presentational classes ››* which are typically pages to be generated. An example of a *‹‹ user interface view ››* is presented in Figure 3-15 and its semantics in Figure 3-14.

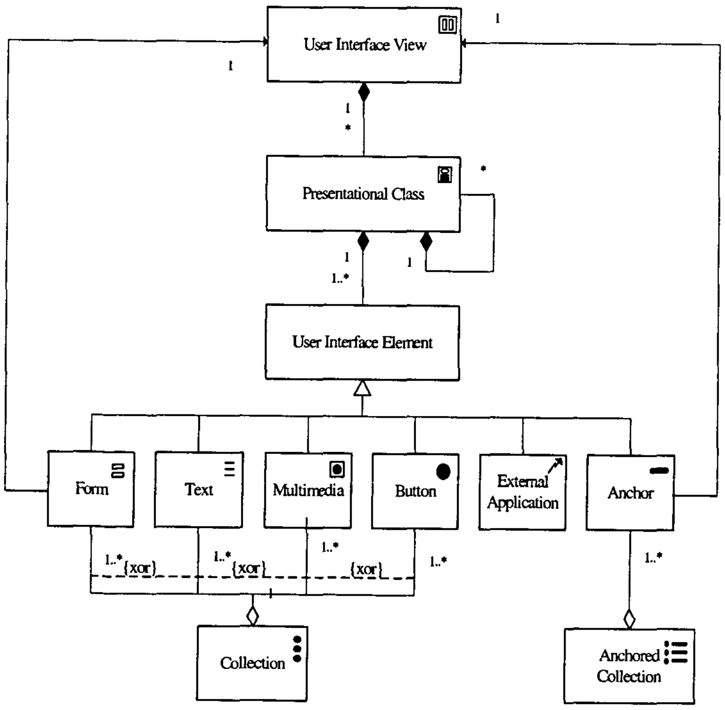


Figure 3-14 Metamodel of the User Interface

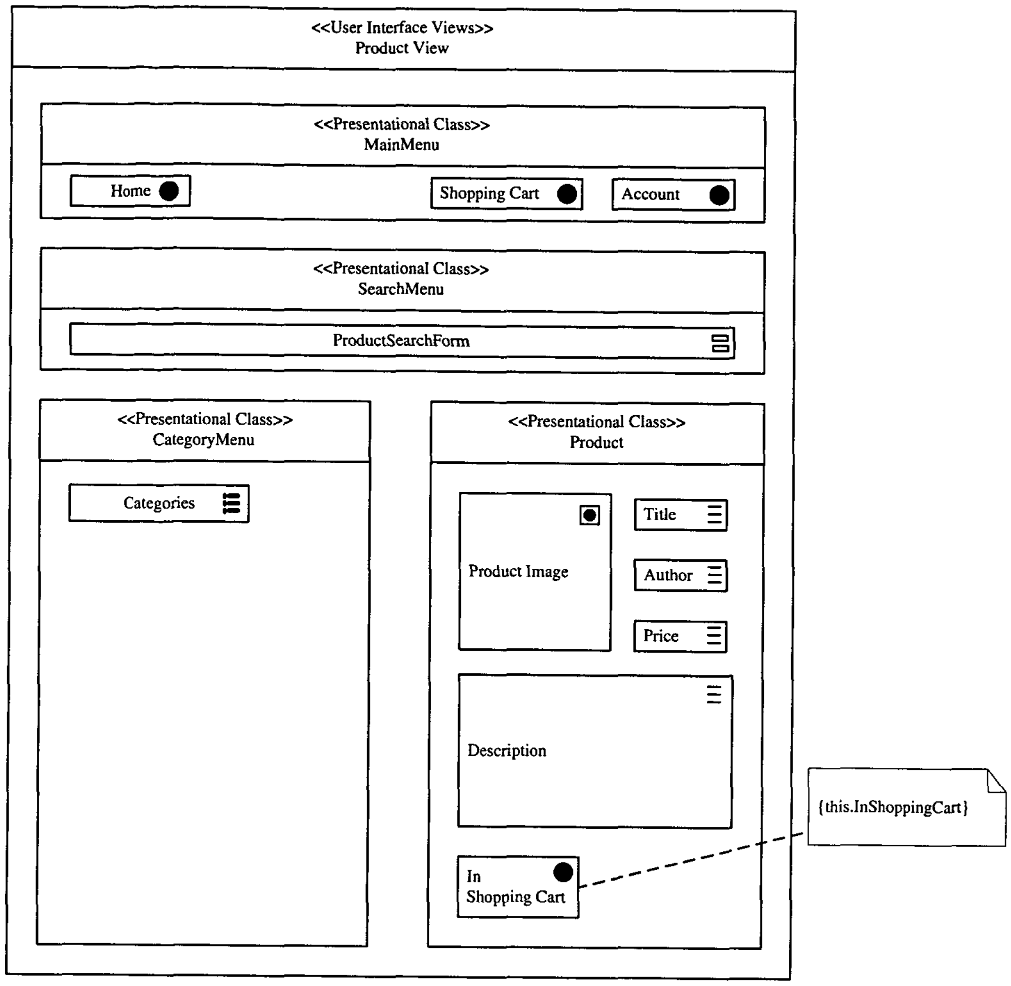


Figure 3-15 Example Of A User Interface

3.6.1.4 Storyboard Scenario

User interface views are very helpful in the communication between the web designer and the customer. It not only gives the customer a clear vision of the presentation, but it also provides a means to define the user interface.

By connecting the anchors in a user interface view with other user interface views, the possible navigation flows are shown. In this way sequences of user interface views are modelled. The models of these sequences are called *storyboard scenarios*. These scenarios give the customer an idea of the navigational flow of the application. An example of a storyboard scenario is shown in Figure 3-16.

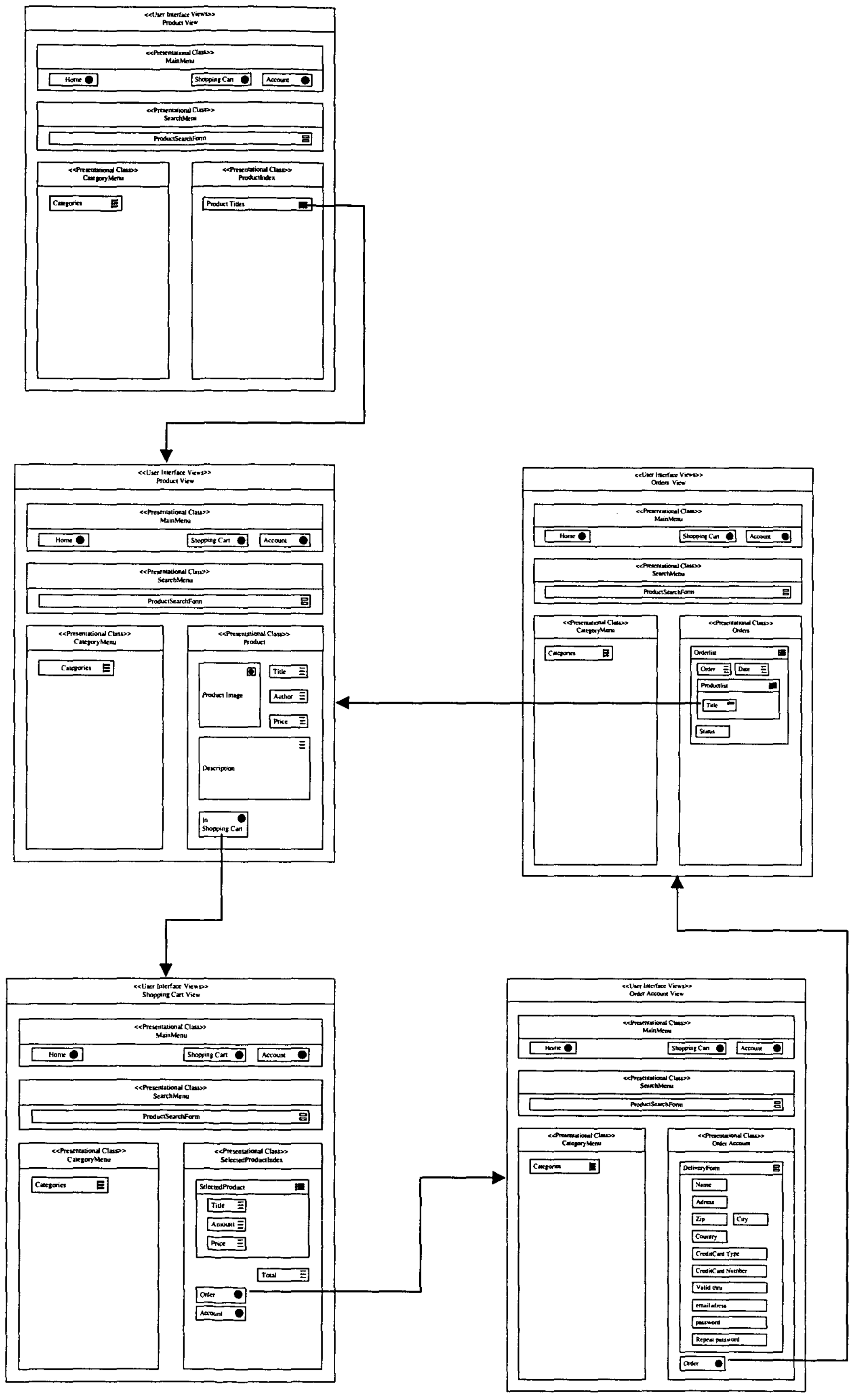


Figure 3-16 An example of a Storyboard Scenario

3.6.2 THE STATIC PRESENTATIONAL MODEL

The static presentational model can be build form a combination of the sketches and storyboard scenarios from the previous section and the navigational structure model. The presentational model defines where and how the navigational objects and access elements will be presented to the user in terms of frames and windows. So the designer has to decide whether he uses frames and framesets and how many and whether he uses multiple windows.

Furthermore he has to specify what the targets of the navigational links are. When there is only one window and one frame, the result is the storyboard of the previous section. Each click gives the next user interface view in the sequence represented by the storyboard. But when there are multiple windows and framesets, the presentation is more complex.

3.6.2.1 MODELLING ELEMENTS

Just as the storyboard the presentational model uses user interface elements. But the presentational model has some additional modelling elements, that are also modelled by UML stereotyped classes and associations. These stereotypes are originally defined by Conallen in [CONA99], but there are some differences. The stereotyped classes are:

- *‹‹ window ››*
- *‹‹ frameset ››*
- *‹‹ frame ›*
- *‹‹ client page ›››*
- *‹‹ server page ››*
- *‹‹ client page ››*
- *‹‹ server script ››*
- *‹‹ client script ››*

The stereotyped associations are:

- *‹‹ builds ››*
- *‹‹ redirects ››*
- *‹‹ submits ››*
- *‹‹ links ››*
- *‹‹ displays ››*
- *‹‹ targets ››*

The stereotype *‹‹ window ››* models the concept of a window, which is used to display the presentational objects. The stereotype *‹‹ frame ››* models the concept of a frame, in which a page can be loaded. These two stereotypes are used to specify the location of the presentational objects. The stereotype *‹‹ frameset ››* is an abstract class that is used for composition of frames and other nested framesets, see Figure 3-17.

The stereotypes *‹‹ server page ››* and *‹‹ client page ››* are both modelling elements that model a web page. A client page is generated from a server page. To denote this association the stereotyped association *‹‹ builds ››* is introduced. A *‹‹ server page ››* contains a number of server scripts, written in some server side scripting language like ASP, which generate the server page. A ‹‹ *client page* ›› can contain client side scripting languages like JavaScript. Server scripts and client scripts are modelled by ‹‹ *server page* ›› and ‹‹ *client page* ››, see Figure 3-17

Besides client scripts, a ‹‹ *client page* ›› may also contain user interface elements. The user interface elements are the same as the ones that are used in the storyboards. The stereotype ‹‹ *anchor* ›› is associated with the abstract class ‹‹ *target* ›› by the directed association ‹‹ *targets* ››. The ‹‹ *target* ›› is a generalization of ‹‹ *frame* ›› and ‹‹ *window* ››, as these are the modelling elements that can be targeted. The target is associated with an abstract class ‹‹ *web page* ›› by the directed association ‹‹ *displays* ››. The ‹‹ *web page* ›› is the generalization of ‹‹ *client page* ›› and ‹‹ *server page* ››. The ‹‹ *anchor* ›› is associated with the ‹‹ *web page* ›› through the directed association ‹‹ *links* ››. The ‹‹ *web page* ›› can be associated with itself through a directed association ‹‹ *redirects* ››, see Figure 3-17

The user interface element ‹‹ *form* ›› can be associated with a ‹‹ *server page* ›› by the directed association ‹‹ *submits* ››. This association is used to model the action to be taken when the form is submitted, see Figure 3-17.

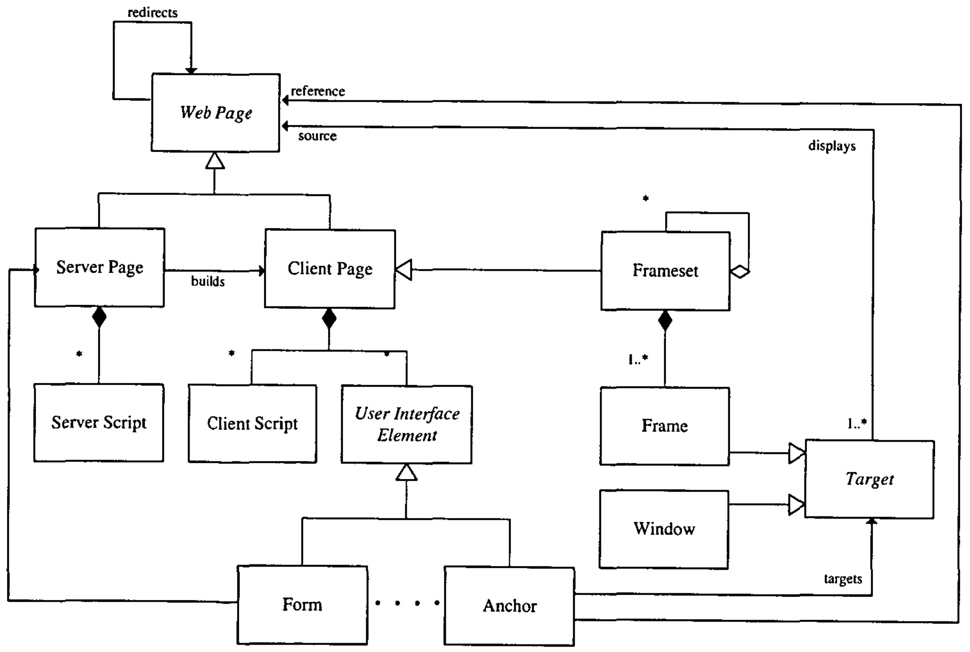


Figure 3-17 Metamodel of the Modelling Elements of the Presentational Model

3.6.2.2 PRESENTATIONAL MODEL

The presentational model is build from the composition of the modelling elements and the navigational structure model. The modelling elements are integrated into the navigational structure model, according the composition structure of the metamodel in Figure 3-17. The navigational objects of the navigational structure model are replaced by web pages. An example of a presentational model is given in Figure 3-18.

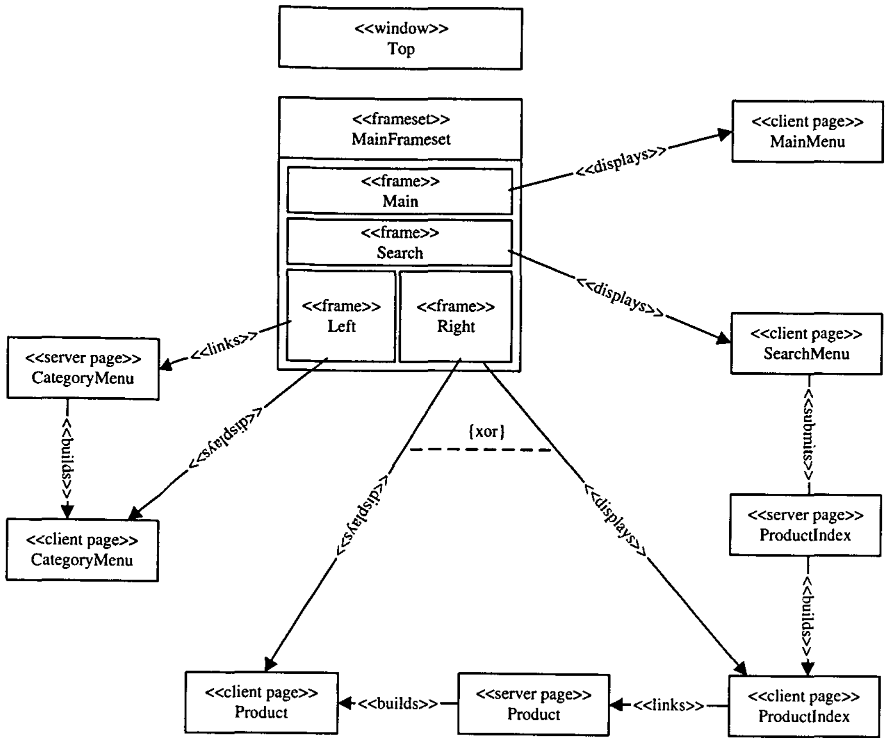


Figure 3-18 Static Presentational Model

3.6.3 THE DYNAMIC PRESENTATIONAL MODEL

The presentational model presented in 3.6.2 is a static model. It does not tell anything about the behaviour and dynamics of the presentation. To model the changes on the user interface when the user interacts with the application, a dynamic presentational model is used. This dynamic presentational model is also called a Window Flow Model, because it specifies which frameset, frames and pages are displayed in which window at what time.

The dynamic presentational model is a UML sequence diagram. A sequence diagram is a two dimensional diagram. The vertical axis represents the time and the horizontal axis displays a set of objects. As the model specifies the state of the windows, the horizontal axis represents the windows and the framesets, which are modelled by the *« window »* and *« frame »* of Figure 3-17. Figure 3-19 shows a dynamic presentational model for a sequence of navigation for our web shop.

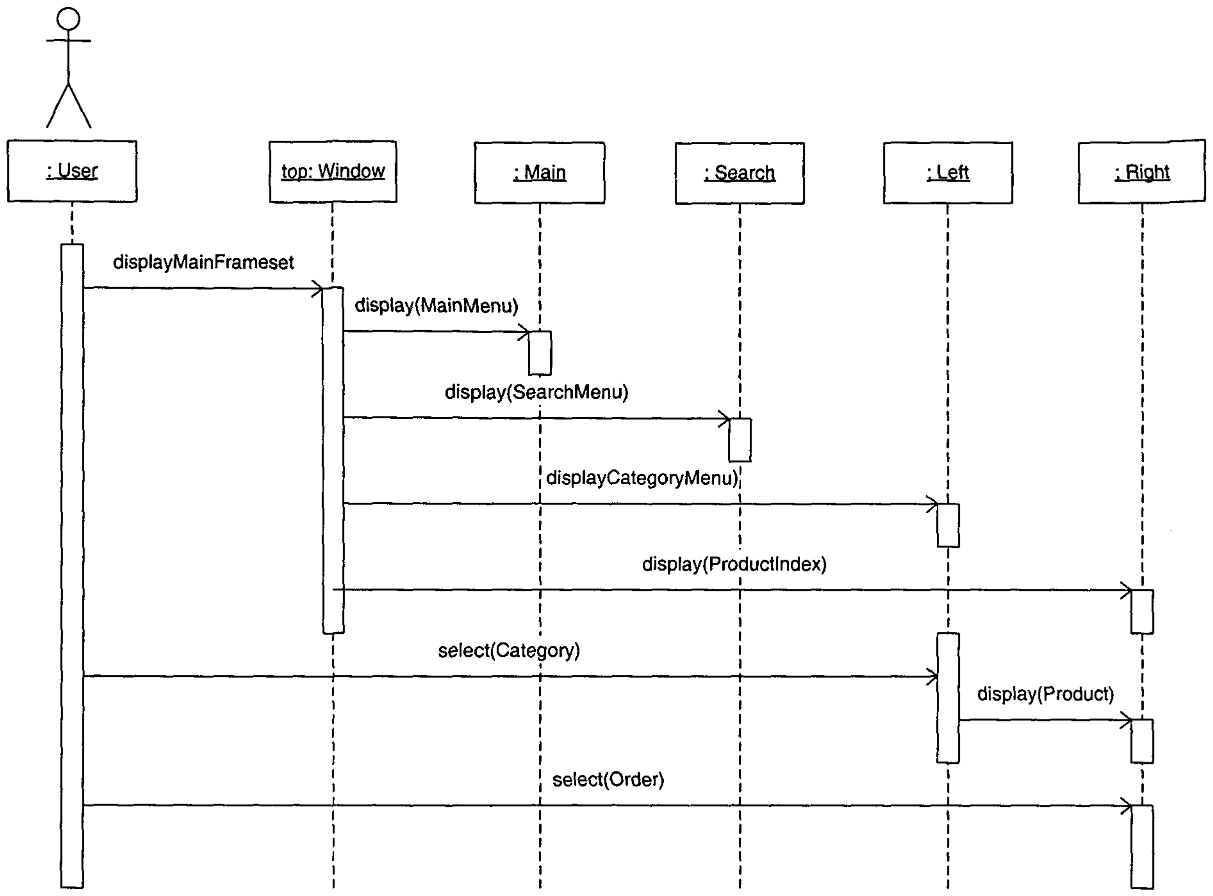


Figure 3-19 An Example Of A Dynamic Presentational Model

4 A METHODOLOGY FOR HYPERMEDIA DEVELOPMENT

4.1 INTRODUCTION

Systems development methodologies are defined to help plan, manage, control and evaluate the development process. By specifying sequences of tasks the development process is organised into controllable parts. But only specifying tasks is clearly not enough. A method should also provide the techniques that are used when performing the tasks. It also should define the deliverables of the tasks, the input needed for the tasks and rules and guidelines of how to perform the task. A sound methodology provides the people involved in the development process a systematic approach to the life cycle of the application.

Hypermedia systems development differs from traditional software systems development in that many people with different skills, such as multimedia designers, programmers and authors, are involved. When developing with such a number of different people, the need for a systematic approach is even more a necessity to keep control over the developing process. But there are not many methodologies that cover the life cycle of a hypermedia system.

In this chapter we present a methodology for hypermedia system development. For each phase of the life cycle, as defined in chapter 2, the tasks to be performed are discussed. But first we define some terms that are needed in describing the phases.

An *artefact* is a piece of information needed in the development process. It is created and used by the *actors* performing a *task.*

An *actor* is a person with certain capabilities to perform one or more specific tasks in the development process. The actor is not the same as a person. A person can play several roles of one or more actors in the development process.

A *task* is a piece of work. Each task has some artefacts as input and output. The artefacts delivered on output are called *deliverables.* A *task* is assigned to an *actor* who has to perform this task.

A *stakeholder* is an actor who is interested in the deliverables of the process. This actor could be someone not directly involved in the development process.

Each phase in the development process consist of a number of *workflows.* A *workflow* is a sequence of coherent tasks which produces one or more deliverables.

The hypermedia methodology is defined by describing each phase in the above terms. In this way a detailed specification of the life cycle is created. In the next sections the phases of the life cycle and its workflows are discussed. Each phase is divided into four sections. The first section describes the workflow. The second section describes the artefacts of the phase. The third section describes the tasks and the actors performing the tasks. The last section is a summary of the previous three.

4.2 Analysis

4.2.1 Workflow

As discussed in section 2.2 the main goal of the analysis phase is to capture the requirements towards the application and get a clear vision of the desired result. Capturing the requirements is probably the most difficult and important task in the development process. Often the development of an application starts with a vague and not a well thought-out idea, either as a solution to solve a problem or maybe as something wanted because the technology is provided. This idea often contains already expectations towards the functionalities of the application.

Hypermedia applications even create more difficulties, because it's a fairly new technology and there's not much experience in the systematic development of hypermedia applications. The technologies and resources provided for hypermedia applications change rapidly and with them requirements, expectations and possible solutions.

To capture the requirements, it is necessary to gather as much information as possible. Information about the functionalities the application should or should not have. But also information about the environment in which the application has to perform its tasks. There are three notions of requirements to considerer:

- Functionality
- Environment
- Expectations

The *functionalities* of an application describe the behaviour of the application. By defining what the application should do and what it should not to do, the behaviour can be described in an abstract way. And by defining what the application should look like and should not look like, the lay-out can be described. There are several kinds of functional requirements, which are already discussed in section 2.3.1. Functional requirements are the main input for the design phase as they tell the designers what has to be designed and what not.

The *environment* is where the application has to perform its tasks. An application is often build with the intention to aid in or automating a working process. So when trying to capture the requirements, it should not be overlooked what the application is meant to do and certainly not which and how people have to work with it. Other environmental requirements are those toward performance and environmental constraints. Environmental requirements are also input for the design phase, but are also of importance when choosing which the techniques to use in the implementation phase.

The *expectations* towards an application are perhaps the most difficult requirements to capture. As the functional requirements describe what the application should and should not do, capturing the expectations is more about what the application can and cannot do. A customer often does not know what the technical possibilities are or even the other way around. The customer thinks something is possible but is not possible because when some features are implemented other things are not possible. Management of expectations is a very important task in preventing disappointment from a customer about the result. Capturing expectations has nothing to do with design choices or implementations, but are necessary to make a well founded decision to continue with a project or not.

Management of expectations is not the only thing needed to make a sound decision. When a clear vision of the desired result is obtained, both a risk and a cost analysis are made. Both analyses are obvious necessary to be able to make such a decision. Also a rough project planning is made.

4.2.2 THE ARTEFACTS

The main goal of this phase is to capture the requirements imposed on the application by the customer. So the most important artefact in this phase is the requirements specification. This specification can be divided into the following artefacts:

- *Work/Information flow*
  This is a description or model of the task the application has to perform. It includes a description of the users involved and a description of the working processes the application has to replace or support. This description gives a clear understanding of the business domain problem.

- *Use Case Model*
  The Use Case model is a model of the functional requirements of the application. The users and working processes identified in the work/information flow are modelled and defined in a detailed way. This Use Case model is the blue print for the design phase.

- *Storyboard scenarios*
  Scenarios describe sequences of actions performed by actors or the application. The storyboards are used to show the elements presented to the users. Together they provide a clear vision of the navigation and presentation of the application.

- *Drawing Sketches*
  As discussed earlier, hypermedia applications often have to satisfy corporate lay-out regulations and rules regarding presentation. Drawing sketches are made to give an impression of the look and feel of the application. Often together with these sketches a document is provided in which the fonts, colours, etc. of the application are defined.

- *Environmental requirements*
  Environmental requirements are non functional requirements imposed on the application like performance and environmental constraints. Sometimes there are corporate regulations and rules regarding to software or implementation techniques to use. These requirements can be recorded in the form of documents which elicit these requirements.

- *Expectations*

As discussed before, Management of expectations is a very important task in preventing disappointment from a customer about the result. So documenting the expectations is a helpful way to keep the expectations in mind when designing and developing the application.

- *Project Plan*
  The project plan is a detailed planning of the whole project, including a time schedule. This time schedules consist of the dates deliverables have to delivered, the milestones and their deadline, and the people involved. It is a blue print for the developing process.

4.2.3 THE TASKS AND THE ACTORS

As can be told from the number and diversity of the deliverables of the analysis phase, already a lot of different actors are involved in the analysis phase. Each actor has one or more specific tasks to perform. However, we will not elicit each actor and the skills needed, but present the tasks in a chronological way, together with the actor or actors that perform the task. In this way the tasks and actors are presented in a more detailed presentation that suits the workflow of the previous section.

The main goal of the analysis phase is to capture the requirements. The analysis phase starts with an inventory of the demands, wishes and expectations of the customer. This is done by a business expert, whose expertise is modelling the organisation. The business expert identifies the actors, use cases and their relationships, called the business domain and produces a work/information flow model.

Next to the functional requirements, the environmental requirements and expectations are documented. All requirements towards performance, security, design, software and hardware as well as budget and time constraints are discussed. These requirements are not always imposed by the customer. Sometimes required functionalities impose other requirements towards the application, for example browser versions.

The business expert usually does not have this kind of expertise. The evaluation of the environmental requirements is performed by a software architect. The software architect is expert in the construction of software, usually responsible for the technical aspects of the development process. The architect provides the business expert with information about technical environmental requirements, which are, together with the budget and time constraints, the main input for a risk analysis. The software architect is also the actor that makes an estimation of the time needed for the development of the application.

The last thing needed, before being able to make an offer, is the evaluation of the requirements towards user interface. As there are two aspects towards user interface, i.e. presentational and navigational, there are two actors involved. The information engineer is an expert in information modelling. The engineer knows everything about presenting and accessing information, such as navigating and searching information. The presentational aspects are evaluated by the user interface designer, whose expertise is look and feel and corporate lay out.

The information gained and evaluated are the input for a cost and risk analysis performed by the business expert. The following information is now available for the business expert:

- What needs to be done?
- How needs this to be done?
- How long will it take?
- What the risks are?

Together this provides the business expert enough information to make the customer an offer, in which all risks are covered. When this offer is accepted, the project continues with the capture of requirements, only now in a more detailed way.

The business domain described in the work/information flow model, is now translated into a Use Case model. The business expert and the customer provide the software architect and the information engineer with the necessary information to construct a Use Case model. This Use Case model is used as the input for a detailed project planning. The Use Cases are structured by the software architect into controllable pieces and put in a certain order in which they will be designed, developed and implemented.

During the construction of the Use Case model, storyboards scenarios and drawing sketches are made. The storyboard scenarios are constructed by the business expert and the customer. The user interface designer designs drawing sketches of the application according to the requirements imposed on the presentation.

The Use Case model, the drawing sketches, storyboard scenarios and the detailed project planning are the blue print for the development process. All three are very intuitive methods for defining the functionalities as they can be well read and understood by a customer. They have to be validated by the customer and can be seen as a milestone. When the customer has approved, the project can move on to the phase: the design.

4.2.4 SUMMARY

Analysis				
Workflow	Tasks	Deliverables	Actors	Stakeholders
Intake & Offer	Project intake	Specification of wishes, demands, expectations of the customer	Business Expert	
	Business domain modelling	Work/information flow	Business Expert	
	Evaluation of functional and environmental requirements	Time estimation and technical requirements	Software Architect UI Designer	Business Expert
	Risk and cost analysis	Offer	Business Expert	Customer
Milestone	Validation of the offer by the customer			
Requirement analysis	Use Case Modelling	Use Case Model	Software Expert Information Engineer	Business Expert
	Storyboarding	Storyboard scenarios	Business Expert Customer	
	Sketching	Drawing Sketches	UI Designer	Business Expert
	Planning	Project Plan	Business Expert	Customer
Milestone	Validation of the Use Case Model, Storyboard Scenarios, Sketches and Project Plan			

4.3 Design

In the *design phase* the application is designed and modelled. The behaviour of the application and its environment is already described in the requirements specification delivered in the analysis phase. So this specification is taken as a starting point for the design phase. As discussed in section 2.3.1 the first step in the design phase is to make a functional design.

The functional design is defined as a detailed document in which all required functionality and the look and feel of the application are precisely defined. As discussed in 2.3.1, the functional design is also done in the analysis phase. The requirements specification describes the functionality in terms of concepts. The functional design defines these concepts defined in a detailed manner. The Use Case model produced in the analysis phase provides, together with the drawing sketches and storyboard scenarios, a precise and complete description of the functional requirements, and can be seen as the functional design. So we start this phase with the technical design.

4.3.1 Workflow

Technical design starts with a conceptual model defining the classes and objects of the problem domain. The conceptual model only considers the objects and classes defined in the functional design. It models the properties of these objects and classes and the relationships between the objects and classes. This is often referred to as a data model. It is an abstraction of the data and the relationships between different types of data. The conceptual model is also taken as a starting point of the navigational and presentational design.

The navigational design is an often overlooked design step. It requires not only technical knowledge, but also awareness of cognitive issues. Not only the way how information is displayed, but also how it can be accessed is a very important issue. As discussed in section 3.5 navigational design consists of two models: the navigation space model and the navigational structure model.

The navigation space model defines the space in which the information is presented to the user. It represents the way objects can be visited when navigating the application, and is a sub graph of the conceptual model. The objects and associations not belonging to this sub graph are not visible to the user but are present in the background. The navigation space model is taken as the starting when constructing the navigational structure model.

The navigation space model represents which objects can be visited when navigating the application. The navigational structure model represents how these objects can be visited. In section 3.5.2.1 navigational context is defined. The notion of navigational context is introduced to specify which nodes in the navigation space are made available for the user. These navigational contexts can be accessed in several ways, for example by indexes or menus, so access elements are defined. With the aid of navigational contexts and access elements a navigational structure model can be designed.

The navigational structure model represents how information can be obtained. But it does not tell us, what this information looks like. This is done during presentational design. The presentational design takes the navigational structure model, functional design and the storyboard scenarios of the functional design phase as input. The navigational structure model provides input of the objects that are presented to the user. The storyboard scenarios provide input of the attributes of the objects that are presented and which modelling elements are used for their presentation. The functional design provides the input of the functionalities of these modelling elements. The method of how to build a presentational model is discussed in section 3.6.

4.3.2 The artefacts

The technical design consists of the following artefacts:

- *Conceptual Model*
- *Navigation Space Model*
- *Navigational Structure Model*
- *Static Presentational Model*
- *Dynamic Presentational Model*

As these artefacts are the design models already discussed in an extensive way in chapter 3, they are only enlisted here.

4.3.3 The tasks and the Actors

The tasks that are performed in this phase are the construction of the models enlisted in the previous section. The first model constructed is the conceptual model. This model is an abstraction of the objects and their relationship and is often called a data model. It is constructed by either the software architect or the information engineer. It is however the responsibility of the software architect, because it is the foundation of the rest of the design. A poor designed conceptual model leads to a poor navigational and presentational design, which leads to a poor implementation.

Based on this conceptual model, the information engineer constructs a navigation space model. The engineer models the navigation space according to the functional design. He then identifies the navigational contexts in the space model. By adding access elements, like menus and indexes, the navigational structure model is constructed. This model defines the navigational structure of the application and is the foundation for the presentational design, because every navigational object and access element has a presentation.

As the static presentational model defines how the structure is presented to the user, it is defined by two actors. The information engineer and the user interface designer. The information engineer first defines how the structure is technically presented to the user in terms of frames, windows, server script pages, etc. The user interface designer defines the sizes, colours, fonts, etc. They also construct the dynamic presentation model together in the same way.

Verification and validation is also already done on delivering the functional design. The functional design is the foundation for the other design steps and implementation. So only after an approval of the functional design of the customer, the design phase is continued. A second verification and validation is done after finishing the design phase. This time the technical and presentational designs are validated.

Often the designs do not have to be validated by the customer, because the technical background of the customer is not enough for a good understanding of the technical design. But even then, there could be still technical design choices that are of relevance for the customer, such as the extensibility of the application. So an approval of this kind of design choices would be recommended. The approval of the presentational design on the other hand is actually already done by validating the storyboards and drawing sketches.

4.3.4 Summary

Design				
Workflow	Tasks	Deliverables	Actors	Stakeholders
Conceptual Design	Define data model	Conceptual Model	Software Architect	Information Engineer
Navigational Design	Define Navigation Space	Navigation Space Model	Information Engineer	
	Define Navigational Context Define Navigational Structure	Navigational Structure Model	Information Engineer	UI Designer
Presentational Design	Define UI Views Define Presentational Classes Define Static Presentation	Static Presentational Design	Information Engineer UI Designer	
	Define Dynamic Presentation	Dynamic Presentational Design	Information Engineer UI Designer	
Milestone	Validation & Verification of the technical & presentational design			

4.4 Realisation

4.4.1 Workflow

The realisation phase is the phase in which the application is physically built according to the technical design. First a choice of the implementation techniques is done. This choice is often already done during the design phase, because requirements limit the number of available choices. Implementation is often done in all phases. In the analysis phase to construct some sort of prototype. In the design phase to create architectural models. In the introduction phase to solve errors and in the maintenance phase to do updates.

When implementation has taken place, the application is subjected to a number of tests. In section 2.4 we already discussed a number of tests:

- Unit tests
- Integration tests
- Functional tests

Unit tests are performed after implementing a component. During the unit test the components it is checked whether the component is error proof and meets its required functionalities.

Integration tests are performed after having finished several components and assembling them into subsystems. During the integration test it is checked whether the integrated components are functioning correctly. The last test performed in the realisation phase is the *functional test.*

Functional test are performed to check whether the implemented system meets the required functionalities as they are defined in the functional, technical and presentational design.

All three tests are performed in the development environment using the test documents produced in the design phase. These test documents can also be used in the next phase when the customer will test the application. When the functional test is successful the application is ready to be installed by the customer and the development process moves onto the introduction phase. When the test fails, the errors and shortcomings are fixed and all tests are performed again. This iteration is performed until the tests are successful.

4.4.2 ARTEFACTS

During the implementation phase, the following artefacts are constructed:

- *Content*
  Content, like images, txt, etc., is provided by the customer or another content provider. This content is needed for implementing the application. Delivering the content too late, often leads to a postponement of the release of the application.

- *User interface*
  The user interface is the implementation of the presentational model. First the images, buttons, video, etc. are created. These elements are used to construct a user interface which implements the structure and lay-out of the presentational model.

- *Database*
  The database is implemented from the data model, i.e. the conceptual model. If there was no requirement imposed on the type of database, the choice is made here.

- *Implementation*
  The implementation is the creation and integration of the components of the applications. The user interface is extended by dynamic page generations to implement functionalities. The components are put together into a working application.

- *Deployment plan*
  The deployment plan describes the steps to be taken to a correct integration of the application into the environment it has to perform its tasks in. The plan consists of installation scripts, which have to be run to install the application, and installation documents which describe two this installation scripts have to be performed.

- *User manuals & training documentation*
  User manuals and training documentation are needed for a successful introduction of the application. Together with an amount of training, they provide the end users with the necessary knowledge to handle the application.

- *Testing*
  There are three different kinds of tests: unit tests, integration test and functional tests. All these tests have their test documents, which are usually checklists of functionalities and requirements to be checked. Whenever a test fails, an iteration is done in the implementation phase until the test provides a successful outcome.

4.4.3 THE TASKS AND THE ACTORS

Content is provided by content providers, usually the customer itself. The content provider provides text, style guides, logos, etc. which the hypermedia designer uses to construct images, buttons, animations, etc. The user interface designer uses these elements as input to implement the presentational model into pages, window, frames, etc. that implement the structure and lay-out of the presentational model.

The implantation of the database is usually done by the software architect or the information engineer as these actors also design the data model. It can also be done by a database expert, whose expertise is in implementing data models and databases.

The implementation of the components is done by software developers. Software developers have the skill to implement a technical design into source code. These components are put together by a system integrator, who keeps the overview of the components to be built. The system integrator also performs unit and integration tests whether the components are working according to their defined functionalities.

The deployment plan is written by the software architect, as this actor is responsible for the technical aspects, and the system integrator, who provides input based on the tests he performed.

The user manuals and training documentation is written by trainers specialised in teaching and training of hypermedia application users. They have good communication skills and usually a lot of patience.

The last test to be performed is the functional test. This test is performed by the software architect and the business expert. They check whether the application is working according to the required functionalities. When the test provides a positive result, the application is ready to be introduced.

4.4.4 Summary

Implementation				
Workflow	Tasks	Deliverables	Actors	Stakeholders
Content	Provide Content	Content	Content provider	UI Designer Software Developer
Implementation	Hypermedia elements implementation	Images, buttons, animations, etc,	Hypermedia designer	UI Designer
	User interface implementation	Workable user interface	UI Designer	Software Developer
	Database implementation	Database	Database Expert	Software Developer
	Implementation of the components	Application components	Software Developers	System Integrator
Integration	Testing of the components	Unit test	System Integrator	
	Integrate components	Subsystems	System Integrator	
	Testing integration of the components	Integration test	System integrator	
Deployment	Write deployment plan	Deployment plan	Software Architect System Integrator	
Training & documentation	Write user Manuals	User manual	Trainer	Customer
	Write training Documentation	Training Documentation	Trainer	Customer
Functional test	Testing the functionalities of the application	Functional Test	Software architect Business Expert	
Milestone	Validation & Verification of the documentation and a positive functional test			

4.5 Introduction

4.5.1 Workflow

After a successful functional test in the realisation phase, the application is ready to be introduced in the environment in which it has to perform its tasks. The introduction phase is divided into three sub phases

- Deployment
- Acceptance
- Training

Deployment is the physical installation of the application in the environment it has to perform its tasks in. It is checked whether the application is working correctly in this new environment. This test is not a functional test, but a more general test to check whether it is running.

During this phase a selected group of end users will test the application, which is called the *acceptance test.* This group verifies whether the application matches the requirements and expectations of the end users. Corrections are made as errors and shortcomings occur.

This is also the time in which *training* is given to the end users and in which the *documentation* is corrected and completed. Documents to be delivered include user manuals, training material and final versions of the design documents. These documents are the same as the ones delivered in the realisation and design phase, only now they are corrected and completed.

When the application is *accepted* by the customer, it is *validated.* When validated, the current version of the software and documentation is released. The development life cycle moves on to the maintenance phase.

4.5.2 THE ARTEFACTS

The following artefacts are used in the introduction phase:

- *Deployment*
  Deployment is the physical installation of the application in the environment it has to run. The installation scripts are run according the installation documents produced in the implementation phase.

- *Deployment test*
  The deployment test is performed after the deployment has taken place. It is a global test in which certain functionalities are tested. These functionalities guarantee a working application.

- *Acceptance test*
  After the deployment a selected group of user performs an acceptance test. The acceptance test is the same as the functional test, only now to check the requirements and expectations of the end users.

- *Training*
  A group of end users it trained in handling the application. During this training the knowledge about the application is transferred from the trainer onto these end users.

- *Correcting and completing documentation*
  All documents produced in the design and implementation phase are revised according to the adaptations made after the acceptance test. So when the acceptance test provides positive results, the documentation is also up to date.

4.5.3 THE TASKS AND THE ACTORS

The first task in the introduction phase is the deployment of the application. The deployment is performed by a installation expert. The installation expert is an expert in platforms, database, etc. He knows how to run the scripts and has the knowledge to solve problems related to the environment the application is installed in. After deployment he performs the deployment test.

After deployment a selected group of end users perform the acceptance test. The users are a representation of all end users, each responsible for testing specific properties and functionalities of the application. This group first receive training in handling the application, provided by a trainer.

The acceptance test will provide errors. As they are fixed, the documentation has to be completed and corrected. These correction are made bý the authors of the documents.

4.5.4 Summary

Introduction				
Workflow	Tasks	Deliverables	Actors	Stakeholders
Deployment	Deployment	Installed application	Installation expert	Customer
	Deployment test	Working application	Installation expert	Customer
Acceptance	Training	Transfer of knowledge	Trainer End Users	Customer
	Acceptance test		End Users	Customer
Documentation	Correcting and completing	All documentation	The authors	Customer
Milestone	Validation of the application, provided by a positive acceptance test			

4.6 Maintenance

4.6.1 The workflow

After the introduction phase the maintenance phase begins. During the maintenance phase the application is adapted to:

- *Changing requirements*
  By using the application, often new functionalities or improvements are required, because the first implementation needs some adaptation.

- *Changing resources*
  The change of content may also lead to new requirements towards the structure of the application. Or external applications supplying resources to the application are changed.

- *Changing environment*
  The environment in which the application performs its tasks is changed, for example it is deployed onto another server.

The development of these requirements is a life cycle of its own. The development goes through all four phases, extending the analysis, design and implementation with the new requirement. This is often the case with hypermedia applications because they usually evolve from small into large applications. These life cycles are often described in change requests, which are a small requirements capture and offer for the development of these requirements.

Maintenance is sometimes also performed by some service level agreement. A certain amount of service is provided to the customer, such as performance checks and error fixing.

4.6.2 The artefacts

The artefacts in the maintenance phase are:

- *Change requests*
  Change request are in fact offers for the adaptation of the application. The change request is produced the same way an initial offer is produced.

- *Service Level Agreements*
  Service Level agreements are contracts in which the amount of service provided to the customer is defined. In these contracts the procedures and type of service are recorded.

4.6.3 THE TASKS AND THE ACTORS

As change request are in fact offers, the actors which produce the initial offer, produce the change requests as well. Sometimes, because the changes are only small adaptations, not the entire analysis and design phase has to be carried out. One important task in the maintenance phase is keeping the documentation up to date and with it the maintainability of the application.

A Service Level Agreement is made by a business expert, specialised in SLA. This business expert supervises the execution of the SLA activities that need to be performed according to the agreement and reports to the customer the activities performed.

4.6.4 SUMMARY

Maintenance				
Workflow	Tasks	Deliverables	Actors	Stakeholders
Change Request	Requirements capture	Change request	Business Expert Customer	
	Analysis, Design. Implementation and introduction	Same as life cycle	Same as life cycle	Same as life cycle
Service Level Agreements	Supervising Service Level Agreement	Service Report	Business Expert	Customer
	Service	Correct working application	Software Architect Information Engineer UI Designer Software Developer	Customer Business Expert
Milestone	Validation of the changes done according to the change request			

5 CONCLUSIONS

In this paper a methodology is presented that covers the whole life cycle of hypermedia system development, whereas most methodologies suited for hypermedia system development only cover part of the life cycle.

The methodology is based on UML modelling techniques, because UML is a widely accepted standard in software development. But hypermedia applications have a number of specific properties, regarding navigation and presentation, which cannot be modelled using standard UML models. UML however does provide extension mechanism to define additional UML models for specific needs. With the aid of these extension mechanisms a UML extension is proposed for the design of navigational and presentational models of hypermedia applications.

Step by step the workflows of the methodology are defined in terms of tasks performed by actors, produced deliverables, stakeholders and milestones in the development process. This way the methodology provides the actors involved in the hypermedia system development process a systematic approach to the life cycle of the application. In a detailed, precise and structured way the application is constructed based on separation of concerns. By distinguishing conceptual, navigational and presentational aspects a correct implementation is made satisfying the requirements captured in the analysis phase.

In Figure 5-1 a graphical representation of the methodology is presented. On the left hand side the analysis and design phase is presented and on the right hand side the realisation and introduction phase. The middle part represent the involvement of the actors. The arrows represent the direction of the development phases.

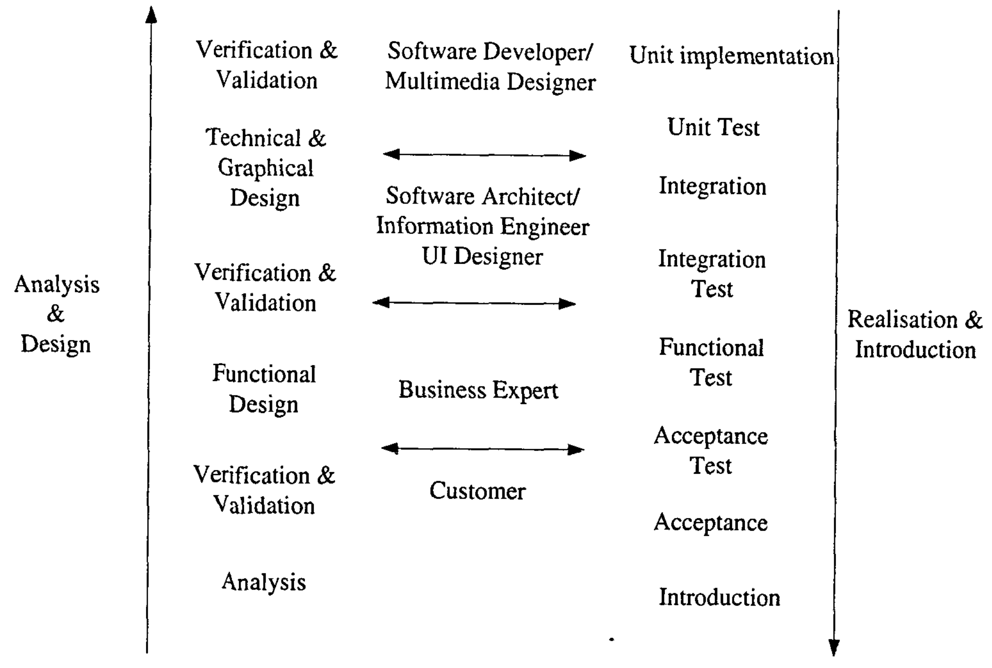

Analysis & Design
Verification & Validation
Technical & Graphical Design
Verification & Validation
Functional Design
Verification & Validation
Analysis
Software Developer/ Multimedia Designer
Software Architect/ Information Engineer UI Designer
Business Expert
Customer
Unit implementation
Unit Test
Integration
Integration Test
Functional Test
Acceptance Test
Acceptance
Introduction
Realisation & Introduction

Figure 5-1 Methodology

A methodology present a systematic approach to a development process. However, every development process is a different one and imposes different requirement on the development process. So the way the methodology should be used is as a blue print for the development process. This way the development process is controlled and the risk avoided of losing focus of the desired result.

6 REFERENCES

[BAUM99] Baumeister, H., Koch, N., Mandel L.: Towards a UML extension for hypermedia design. In Proceedings «UML»'99, France, R., Rumpe, B. (Eds), LNCS, Vol. 1723. Springer-Verlag (1999) 614-629.

[CONA99] Conallen J. (1999). Modeling Web application design with UML. *Communications of the ACM, 10(42), 63-70.* Available at http://www.rational.com/media/whitepapers/webapps.pdf

[GPS93] Garzotto F., Paolini P and Schwabe D. (1993). HDM: A model-based approach to Hypertext application design. *ACM Transactions of Information Systems,* 11(1), 1-26.

[ISAK95] Isakowitz T., Stohr E. and Balasubramanian P. (1995). A methodology for the design of structured hypermedia applications. *Communications of the ACM,* 8(38), 34-44.

[JACO99] Jacobson I., Booch G. and Rumbaugh J. (1999). The Unified Software Development Process. Addison Wesley.

[KOCH00] Koch, N. (2000). Hypermedia Systems Development based on the Unified Process. Available at http://www.pst.informatik.uni-muenchen.de/personen/kochn/UHDP-Methodik.pdf

[LEE98] Lee H., Lee C. and Yoo C. (1998). A scenario-based object-oriented methodology for developing hypermedia information systems, In *Proceedings of 31st Annual Conference on Systems Science*, Eds. Sprague R.

[OMG99] OMG, Unified Modeling Language 1.3, , Object Management Group, 1999

[SCHA96] Schawbe D. and Rossi G. (1996) Systematic hypermedia design with OOHDM. In *Proceedings of the ACM International Conference on Hypertext*, Hypertext '96. Available at http://citeseer.nj.nec.com/schwabe96systematic.html

[SCHA98] Schawbe D. and Rossi G. (1998) Developing hypermedia applications using OOHDM. In *Proceedings of Workshop on Hypermedia development Process, Methods and Models*, Hypertext '98. Available at http://citeseer.nj.nec.com/schwabe98developing.html