Use Cases and Scenarios in the Conceptual Design of Web Applications

Patrícia Vilain
e-mail: vilain@inf.puc-rio.br

Daniel Schwabe
e-mail: schwabe@inf.puc-rio.br

Clarisse Sieckenius de Souza
e-mail: clarisse@inf.puc-rio.br

PUC-Rio Inf.MCC12/00 February, 2000

Abstract: In this work we describe a method for gathering requirements and synthesizing the conceptual design of web applications. Our method emphasizes the importance of modeling user interaction and involving users in this process. It starts by capturing the requirements expressed in use cases and user interaction diagrams, a visual notation we propose to represent user-system interaction. These diagrams are also useful as a requirement validation tool and as input for obtaining the conceptual schema. We show how to derive user interaction diagrams from use cases and how to obtain a conceptual schema from these diagrams. Conceptual design is followed by navigation design, which is a crucial phase for the success of web application development. We finally describe how techniques used in this method can be extended specifically to support the navigational design stage.

Keywords: Hypermedia systems design, use cases, scenarios, conceptual design, requirement gathering

Resumo: O trabalho descreve um método para levantamento de requisitos e projeto conceitual de aplicações web. Este método enfatiza a importância da modelagem da interação entre o usuário e o sistema e o envolvimento do usuário neste processo. Inicialmente os requisitos capturados são expressos através de use cases e diagramas de interação do usuário, notação visual proposta para representar a interação entre o usuário e o sistema. Estes diagramas também são usados como ferramenta para a validação dos requisitos e como entrada para a obtenção do esquema conceitual. O trabalho descreve a derivação dos diagramas de interação do usuário a partir dos use cases e a obtenção do esquema conceitual a partir destes diagramas. O projeto conceitual é seguido, então, pelo projeto navegacional, fase crucial para o sucesso do desenvolvimento de aplicações web. Por fim, é descrito como as técnicas usadas neste método podem ser estendidas para suportar o projeto navegacional.

Palavras-chave: Projeto de sistemas hipermídia, use cases, cenários, projeto conceitual, levantamento de requisitos

INTRODUCTION

The WWW has become one of the major platforms for the implementation of information systems. Together with its architecture, the WWW has also brought forth hypertext (or hypermedia) models, which support the design of how the user "navigates" through information. In fact, current applications exhibit a mixture of navigational and functional behavior, increasing the complexity of the design process. As a consequence, traditional software engineering methods when applied to WWW application design must be enriched to explicitly deal with navigation. In other words, besides conceptual design and the other known phases of the development cycle, such methods need to include a specific navigation design phase [18].

While conceptual design gathers the information and relationships that are meaningful in the application domain, navigation design deals with information content as presented in an application, including all the possible navigation paths that link contents to each other. Whereas traditional requirements elicitation is geared towards synthesizing conceptual and functional models, our goal here is to synthesize models in such a way as to facilitate navigation design as well.

In general, current methods for hypermedia applications design (OOHDM [22], RMM [11], EORM [16], HDM [8]) present both conceptual and navigation design phases. However, conceptual design for them is similar to what it is in general-purpose methods (Unified Process [15], Objectory [13], OMT [20], Booch Method [1], Fusion [4]). Therefore, they are not being particularly tuned to the navigation design phase.

In this paper, we present a method for the requirements gathering and conceptual design of web applications, with a strong emphasis on the characterization of user interactions. The proposed method is based on use cases [13] and scenarios [3], and promotes an intensive involvement of users. We also introduce User Interaction Diagrams as a diagrammatic modeling tool to represent the interaction between the user and the application. We believe that the proposed method, along with our diagrammatic tools, not only helps the designer to obtain a more reliable and complete conceptual schema, but also facilitates the accomplishment of the navigational design phase.

Since web-based applications are inherently interactive, and good web applications must make effective use of the hypertext paradigm [18], special care must be taken during interaction design in order to identify and make good use of navigation opportunities. It must be noticed, in this case, that navigation design involves defining the underlying information architecture the users will navigate in, and in this sense it is fair to claim that it is something other than "pure" interface design.

Consequently, the first important thing in navigational design is to distinguish which interactions will specifically require navigation from those that will require further user interaction design. Interaction is the communication activity that takes place between the user and the application [17], while navigation is the operation of following hypertext links associated to hypertext objects. Navigation is thus one of the possible types of operations triggered by interface operations.

Having defined which interactions are best cast as navigation steps, several aspects must be specified, such as: which collection of information items a current item belongs to; the kind of navigation through data items of a set (sequential, circular, via index); the decision if an index should be used to access a set of data items; the attributes used to represent an index; the ordering criterion of an index; and so forth. We believe that conceptual design should be based on techniques that can be extended and also used in the navigational design. The better the extensions, the easier it will be to accomplish navigational design. The method we propose here heads in this direction, since it allows navigational aspects to be easily incorporated to the models obtained at the conceptual design phase.

This paper includes four additional sections. Section 2 presents our method. Section 3 gives a brief description of relevant characteristics of the navigation design phase. Section 4 discusses related work. And finally section 5 presents our conclusions.

PROPOSED METHOD

In this section, we describe the method we are proposing for requirements gathering and conceptual design. The method is part of the Object Oriented Hypermedia Design Method, OOHDM [22]. Its main phases are shown in figure 1.

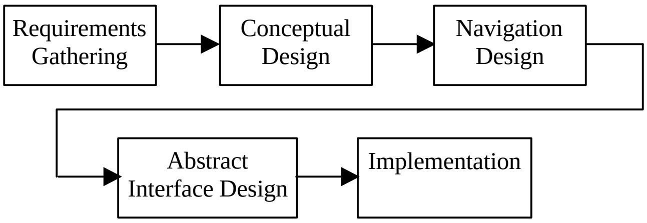


Figure 1 - Phases of OOHDM

Requirements gathering is a new phase added to the original OOHDM, and will use the method and notation proposed in this work. The conceptual design phase produces a model of the application domain. This phase is also being improved in by features presented in this work. Navigation design defines the information content that will be presented in the application and the navigation structures that connects its parts. Section 3.1 presents more details about the navigation design. The abstract interface design phase defines the appearance of the navigational objects and interface objects. Implementation is finally the phase for the mapping of conceptual, navigation and interface objects onto implementation environment elements.

The requirements gathering stage presents the following phases: identification of actors and tasks, specification of scenarios, specification of use cases, specification of user interaction diagrams, validation of use cases and user interaction diagrams. The conceptual design stage, in its turn, is carried out in a single phase: specification of the conceptual schema. Figure 2 shows the phases of the proposed method.

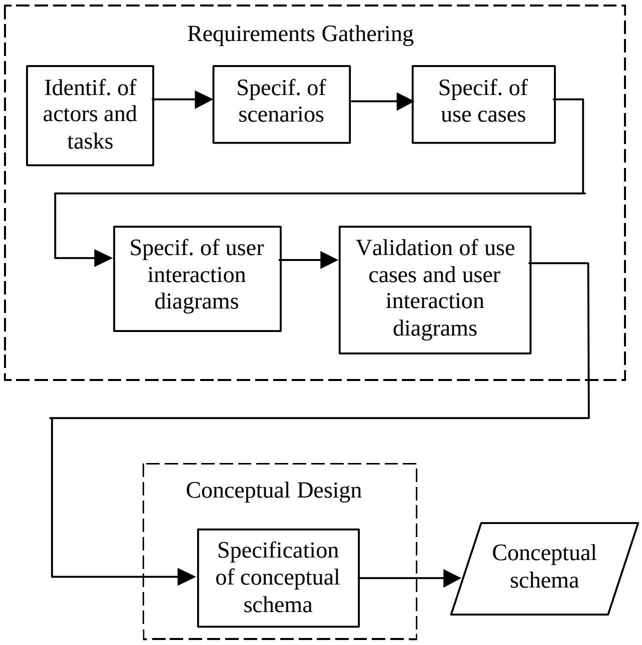

Requirements Gathering
Identif. of actors and tasks
Specif. of scenarios
Specif. of use cases
Specif. of user interaction diagrams
Validation of use cases and user interaction diagrams
Conceptual Design
Specification of conceptual schema
Conceptual schema

Figure 2 - Phases of the proposed method

We applied this method in a case study for CD selling in a virtual shop. A virtual shop must present the same basic functionality as traditional shops. Six people participated in the case study. Five of them regularly navigate the internet and sometimes shop online. One participant has never used the internet and only one participant is familiar with the techniques involved in this method. Participants produced 49 scenarios, from which 14 simple use cases and 4 parameterized use cases were derived by the application designer.

Requirements Gathering

Identification of actors and tasks

In this phase, the designer interacts with the domain in order to identify actors and tasks. This interaction is achieved through analysing documents and interviewing users. The main goal is to perceive and capture users needs.

First, the actors of the domain are identified. Actors are entities that exchange information with the application. An actor represents a specific role of a system user [15]. A user can have several roles, thus representing several actors.

It is important to identify the different types of actors because they generally don't participate in the same scenarios (although this may occasionally not be the case). Thus, only the relevant scenarios are discussed with each user.

An example of actor identified in the CD shopping domain is the *customer*. Customer is someone who buys CDs from a virtual shop. Another example to this domain is the *typist* responsible for entering the data related to the CDs. A user can simultaneously play the customer and typist role.

Secondly, for each type of actor, the tasks the web application must support are also identified in this phase. This identification will be useful when the users have to describe their scenarios.

To exemplify the task identification, some tasks of the type of actor *customer* are shown below:

- *Buying a CD based on the album's title;*
- *Buying a CD based on the title of a song it contains;*
- *Searching the best selling CDs.*

When the name of the task is not enough to guarantee that one will understand it, it is necessary to give a little description of it. If a type of actor or task is incorrectly identified in this phase, it will be corrected during the next phase, namely the specification of scenarios.

Specification of Scenarios

In this phase, the scenarios describing the user tasks are specified. Scenarios are narrative descriptions of how the application may be used [3] and have to include the context in which they occur, their goal and a narrative of the sequence of actions performed [7]. Although they may be produced by users and designers alike (with different purposes), we will concentrate on those produced by users.

Each user specifies the scenarios that describe his tasks. If the user belongs to several classes of actors, the class of actor to which the scenario belongs must be identified. The user can describe the scenario textually or verbally. No matter what is the kind of the description given by the user, the designer should never interfere with it.

In order to help users identify their scenarios, designers can make use of the tasks identified in the previous modeling phase. Such tasks are useful when users don't have any knowledge about scenarios or when they forget some cases of relevant scenarios. In our case study, all users, except one, didn't know how to proceed to describe scenarios. It was then necessary to explain what a scenario is and show them examples. Some users also included the identified tasks earlier in their scenarios, but only when they thought such tasks were important.

In the following, a scenario specified by a user during our case study is presented. This scenario describes a user buying a CD based on the name of the artist.

Scenario: Buying a CD based on the name of the artist.

"I enter the name of the band or its initials (ex. "Ramones" or "Ra") and the system gives me one or more bands whose names match with the given name. Then I select the band that I want and its CDs are presented along with each corresponding cover. If one is the CD I want, I then select the CD and it is added to my shopping basket. It would be interesting if the price of the CDs was informed[1]. I can also buy several CDs if I wish just by clicking over the CDs I want.

Specification of use cases

A use case is a way of using the system [15]. Use cases deal only with the interaction between the user and the system or the information visible to the user. They don't deal with the internal aspects of the system. They also don't mean the same as scenarios do, since they represent an abstraction of the set of all possible scenarios dealing with the same tasks [21]. A scenario represents a specific instance of a use case [14].

Thus, the specification of use cases requires the grouping of all scenarios that have the same function. Scenarios that have the same function are grouped in one single use case. Since a scenario may address more than one task, it may belong to several use cases.

The description of a use case has to include the information presented in all related scenarios. Therefore, the scenarios described by the users can have information that is not relevant, having only a complementary role. So, the designer is responsible for the identification of the relevant information. Note that the information selected by the designer is bound to a set of data items. Therefore, to identify the relevant information means to select which data items are relevant to the user.

In order to specify a use case, we do the following:

- Identify the data items exchanged in the interaction between the user and the application. Generally these data are represented by nouns, but it is important to get only the nouns that have clear and well defined semantics;
- Considering the data items exchanged, identify which are given by the user and which are returned by the application;
- Identify all the data items that are associated between themselves. These have to appear together in the use case text. For example, scenario 1 of user 1 says "*…for each CD, its year, music content, availability and price are presented…*". Therefore, the year, music content, availability and price have to be associated to compose the information about a CD;
- Identify the data items that are organized as a set. These appear in scenarios as a list of items, a characterized set, or they are written in the plural. In the text describing the use case, these data items have to be explicitly referenced as a set.
- The action sequences that appear in scenarios have to appear in use cases as well. If the action sequences of the scenarios are conflicting, it is necessary to define different use cases or talk to the users to resolve the conflict.
- The operations performed against the data items, generally represented by verbs, have to be included in the use cases.
- Since a use case is an abstraction of a set of scenarios, all instances presented in the scenarios must be generalized into the use case.

The designer can also derive new information [5] from that given by users. However, the designer must validate his interpretation with the users in the next phase of the method, namely the validation of the use cases.

In the following figure, the definition of the use case *Select a CD based on the artist's name* is presented. All scenarios that deal with this subject are examined. We present only three of the scenarios that were actually examined. All the information included in the use case appears underlined.

Scenario 1 - User 1: Buying a CD based on the artist's name.

"I want to by a CD of Rolling Stones; I type in the name of the singer and I wait for the list of CD's by the singer I mentioned. For every returned CD, I see the year, the music content, its availability and price. There is a possibility of a more detailed search for CD's; in it, in addition to the information above, I see the titles of the songs, the duration of tracks, the composer's name, the image of the album's cover, and the possibility of listening to a short sample of each song. I select the desired CD; if I decide to buy another one I can query again the information of other CD's and select the one I want to buy. In the end, I can exclude a CD I had selected, select a new one, give up the purchase or confirm it."

Scenario 1 – User 2: Buying a CD based on the name of the artist.

"I enter the name of the band or its initials (ex. "Ramones" or "Ra") and the system gives me one or more bands whose names match with the given name. Then I select the band that I want and its CDs are presented along with each corresponding cover. If one is the CD I want, I then select the CD and it is added to my shopping basket. It would be interesting if the price of the CDs was informed. I can also buy several CDs if I wish just by clicking over the CDs I want.

Scenario 1 - User 6: Buying a CD of the Beatles.

"I enter the name Beatles and I see all the CD's they have recorded. I select the CD that I want and I see all the songs included in the album. If I want to listen to a song or read something about it, I select its name. All CD's should have an indication of their price.

[1] Note that here the user is expressing a requirement or a wish, and not an action in the narrative.

There are situations in which it is better to separate certain action sequences of a scenario in different use cases, so that it can be referenced by several use cases. For instance, since many scenarios deal with the purchasing of a CD, it is better to separate the purchase sequence from the selection sequence. Thus, other scenarios belonging to other use cases that also deal with the purchasing can refer to the purchasing use case, avoiding duplicating that information.

Use Case: Selecting a CD based on a given artist’s name.

Scenarios: 1.1 / 1.2 / 2.1 / 3.4 / 3.8 / 4.1 / 5.2 / 5.5 / 6. 1 / 6.2

Description:

1. The user enters the name of the artist or part of the name. S/He can inform the year or period of the searched CD if s/he wants to.
2. The system returns a set with the names of the artists matching the entry. If only one name of artist matches, the set of the CDs of this artist is directly shown (step 4).
3. The user selects the artist of interest.
4. The system returns a set with the CDs of the artist. For every returned CD, the title, name of the artist, year, price, availability and image cover are shown.
5. The user can access more information about a CD if s/he wants (use case Show CD).
6. If the user wishes to buy more than one CD, s/he selects the CD(s) and adds it (them) to the shopping basket to perform the purchase later (use case Purchase).
7. If the user is interested in buying another CD of the same artist, s/he can return to the step 5.

A list of scenarios is added to the description of the use case to facilitate the identification of the scenarios that belong to it. This list is composed by pairs (X,Y), where X is the identification number of the user and Y is the identification number of the scenario. For example, the first scenario of the user 1 (*Buying a CD based on the name of an artist*) is represented by 1.1.

A use case can also be incremented by information of other use cases or by using patterns [19]. For example, in the specification of the use case *Selecting a CD based on a given artist’s name* it was verified that a user can buy more than one CD. So, the possibility of buying more than one CD was also included in the use case *Selecting a CD based on the album’s name*.

Although the scenarios gathered in the previous phase describe a sequence of actions without problems, the designer should also include exception handling in the use case specification.

Parameterization of use cases

After the specification of all the use cases, they are parameterized. The use cases presenting the same interaction sequence, but with distinct information, can be represented as a single parameterized use case.

Parameterization of use cases facilitates validation. The designer carries on discussions with the users based on parameterized use cases instead of discussing several similar use cases one at a time. When necessary, the designer refers to the particularities of each individual use case.

The use case *Selecting a CD based on a given parameter (artist’s name or composer’s name)* represents the parameterization of the use cases *Selecting a CD based on a given artist’s name* and *Selecting a CD based on a given composer’s name*.

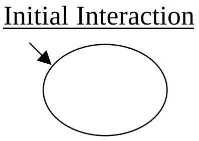
Use Case B: Selecting a CD based on a given parameter
Parameters: artist’s name or composer’s name
Description:

1. The user enters the name of the parameter or part of the name. S/He can inform the year or period of the searched CD if s/he wants to.
2. The system returns a set with the names of the parameter instances matching the entry. If only one instance matches, the set of the CDs of that is directly shown (step 4).
3. The user selects the instance of interest.
4. The system returns a set with the CDs of that instance. For every returned CD, the title, name of the artist, year, price, availability and image cover are shown.
5. The user can access more information about a CD if s/he wants (use case Show CD).
6. If the user wishes to buy more than one CD, s/he selects the CD(s) and adds it (them) to the shopping basket to perform the purchase later (use case Purchase).
7. If the user is interested in buying another CD, s/he can return to the step 5.

Specification of user interaction diagrams

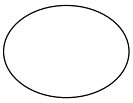
Besides the textual description, we propose a graphical representation for the use cases by what we call **user interaction diagrams**. A user interaction diagram represents the interaction between the user and the application textually described in a use case. Note that this diagram describes only the exchange of information between the system and the user, without considering specific user interface aspects.

User interaction diagrams are built using the following notation:



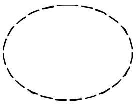
Represents the beginning of the interaction between the user and the application.

Interaction



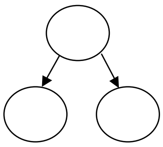
Represents one interaction between the user and the application. The information given by the user and returned by the application is shown inside the ellipse.

Optional Interaction



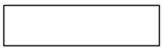
Represents an interaction that depends on the previous interaction. According to the previous result this interaction may occur or not. If it does not occur then the output of the previous interaction will be the input of the next interaction.

Interaction Alternatives



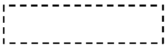
This representation is used when there are two alternative outputs from an interaction. The subsequent interaction depends on the elements or operation chosen by the user.

Data Entry



Represents mandatory data entered by the user.

Optional Data Entry



Represents optional data entered by the user.

Element and its data items

Element (data items)

Represents an element and its data items. The data items are optional.

Element Set

… Element (data items)

Represents an element set. The data items associated to the element are also presented.

Specific Element

Element X

Represents the specific element selected or entered by the user in the previous interaction.

Text

XXXX

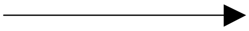
Represents some additional data that participates of the interaction.

Optional Text

XXXX *

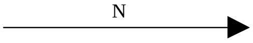
Represents some optional data that participates in the interaction.

New Interaction



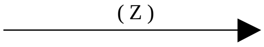
Represents a new interaction that occurs after the user has entered the required data and the application has returned other information in the previous interaction.

Selection of N Elements and New Interaction



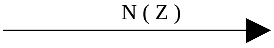
Represents that N elements must be selected prior to the new interaction.

Call of Operation Z and New Interaction



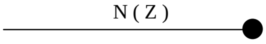
Represents that the operation Z must be called prior to the new interaction.

Selection of N Elements and Call of Operation Z and New Interaction



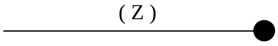
Represents that N elements must be selected and the operation Z must be called prior to the new interaction.

Selection of N Elements and Call of Operation Z



Represents that N elements can be selected and the operation Z called.

Call of operation Z



Represents that the operation Z can be called. The calling is optional.

Figure 3 shows the user interaction diagram of the use case *Selecting a CD based on a given singer's name*.

The ellipses represent the interaction between the user and the application. The interaction begins with the ellipse that has the arrow which is not connected to a source. In this example the user must enter the artist's name (rectangle with a continuous line), while the year is optional (rectangle with a dashed line). Since the next interaction is optional we have two possibilities of interaction. If only one artist matches the artist's name entered by the user then the optional interaction is skipped and the next interaction occurs, thus showing the set of CDs related to the matching artist. If several artists match with the artist's name then the optional interaction occurs, then showing a set of artists and requiring the user to select only one artist. After that, the application shows the set of CDs related to the selected artist. For each CD the following data items are shown: title, artist's name, year, price, availability and image cover. The line with the text *1..N (add to the shopping basket)* means that 1 to N CDs can be selected and added to the shopping basket.

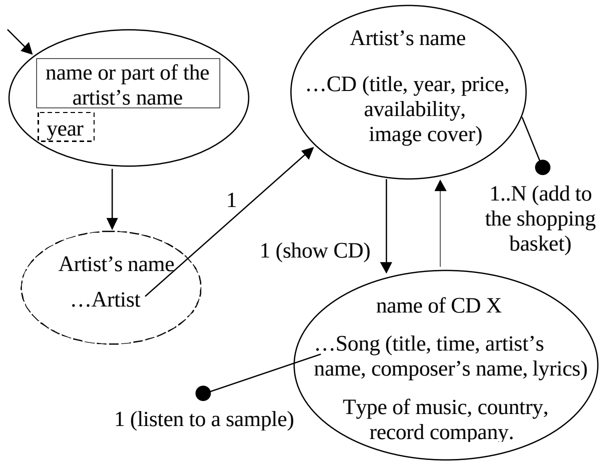

name or part of the artist's name
year
Artist's name
…CD (title, year, price, availability, image cover)
1
1..N (add to the shopping basket)
1 (show CD)
Artist's name
…Artist
name of CD X
…Song (title, time, artist's name, composer's name, lyrics)
Type of music, country, record company.
1 (listen to a sample)

Figure 3 - User interaction diagram

A user interaction diagram is also specified for each parameterized use case. Figure 4 presents the user interaction diagram for the parameterized use case:

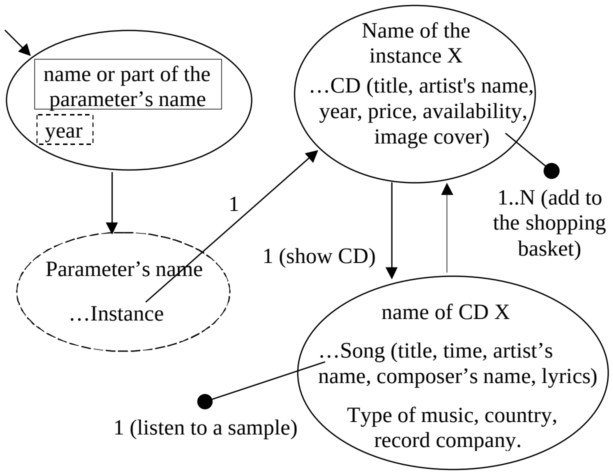

name or part of the parameter's name
year
Name of the instance X
…CD (title, artist's name, year, price, availability, image cover)
1
1..N (add to the shopping basket)
1 (show CD)
Parameter's name
…Instance
name of CD X
…Song (title, time, artist's name, composer's name, lyrics)
Type of music, country, record company.
1 (listen to a sample)

Figure 4 - User interaction diagram for a parameterized use case.

Validation of the use cases and user interaction diagrams

In this phase the designer interacts with each user to validate the use cases and user interaction diagrams. In order to do this the designer shows the use cases and their user interaction diagrams to verify if the users agree with them. If some use cases are parameterized then the resulting use case is discussed instead of the original use cases.

Each user validates only the use cases given by her/him and user interaction diagrams with the roles of actors s/he plays. During the validation of each use case and its user interaction diagram, the designer has to reference the scenarios of that user related with the use case. However, since the use cases contain information supplied by all users, it is possible that the use case being validated does not generalize any scenario described by that user or contains information not supplied by that user. In this case, the designer may ask the user if s/he agrees with the additional information gathered from other scenarios.

It is also important to explain the notation of the user interaction diagrams to the users. We observed that this notation is simple, since in our case study all users understood the meaning of diagrams without difficulty.

A table is used to track the modifications resulting from the use case validation with the users. Each column represents an interaction during the validation with a user. Each row represents a use case (and its user interaction diagram) validated with a user in a specific interaction. For each use case validated in an interaction, a tag records if the user agreed with it (OK), if it was modified by the user (X), if it was inserted in that interaction (I), if it was deleted in that interaction (D) or if it was not validated (-). A use case cannot be validated before being inserted, or after its deletion, or if the user validated another use case that parameterizes it, or yet if it was not modified since the last interaction with that user.

All parameterized and simple use cases must be inserted in the table because in any interaction a parameterized use case can be deleted. In this case, the original use cases will be validated instead of the parameterized use case.

Before beginning the validation with the next user, the use cases are modified according to the validation result of the previous user. For that reason, the ordering of the interactions with the users is recorded in the table.

Table 1 shows a sample of the table of modified use cases produced in our case study. In this table, 9 use cases were validated with 4 users. It registered 2 interactions with the user 2, 3 interactions with the user 3, 2 interactions with the user 1, 2 interactions with the user 4 and one intervention of the designer.

The intervention of the designer occurs when a specific modification in a use case and its user interaction diagram must also be propagated to several use cases. In this case, the designer does all the necessary modifications in the use cases before beginning the validation with the next user.

When a user gives a solution to a problem that is common to several use cases, those use cases and their user interaction diagrams are modified to incorporate the given solution. Therefore, in order to facilitate the validation with the users, only one of those use cases must be validated with the other users since that solution will be adopted in all remaining use cases. For example, during the validation of use case 7 with the user it was verified that the record company is one of the data items associated with CDs. Thus, the record company was added to others use cases. Note that those use cases did not need to be validated again.

The interactions with the users continue until an agreement is reached. If that does not happen, then the designer is forced to choose the solution that s/he considers to be more satisfactory, otherwise more than one application must be defined. When the development time of the application is strict, the number of interactions may be previously determined to avoid compromising the schedule.

Table 1 - Modified Use Cases

Uses Cases Modificados pelos Usuários										
	U2	U3	U1	U4	U3	Des	U3	U2	U1	U4
Use Case 1	x	ok	ok	-	-	-	-	-	-	-
Use Case 2	x	x	ok	ok	-	-	ok	ok	-	-
Use Case 7	x	ok	ok	x	ok	-	-	-	ok	-
Use Case 11	ok	ok	x	x	ok	-	-	ok	ok	-
Use Case 12	ok	ok	ok	ok	-	-	-	-	-	-
Use Case 13	ok	ok	ok	ok	-	-	-	-	-	-
Use Case 14	I	ok	ok	ok	-	-	-	-	-	-
Use Case A	-	ok	ok	x	ok	x	x	ok	ok	ok
Use Case D	-	E	-	-	-	-	-	-	-	-

Conceptual Design

Specification of the conceptual schema

In this step, the conceptual data schema is defined according to the following rules:

1. For each user interaction diagram define a class for each element, set element and specific element. For each defined class, assume the existence of an identifier attribute OID.
2. For each data item of the set elements that appears in each user interaction diagram, define an attribute according to the following:
   - Verify if the data item is functionally dependent[2] on the attribute OID in each class, i.e., if OID → data items. Verify if the data item is not transitively dependent[3] on the OID. If these conditions are satisfied, then the data item must become an attribute of the class. Preferentially, the verification must begin in the class that represents the set because there is a higher probability that this data item will be an attribute of this class.
3. For each data item entered by the user, represented by an unique rectangle, define an attribute according to the following:
   - Verify if the data item is functionally dependent on the attribute OID in each class, i.e., if OID → data items. Verify if the data item is not transitively dependent on the OID. If these conditions are satisfied, then the data item must became an attribute of the class. Preferentially, the verification must begin in the class that represents the set because there is a higher probability that this item data be an attribute of this class.
4. For each attribute that appears in a different set from its class, define a relationship between its class and the class representing the set. It is also necessary to verify if this relationship is semantically correct (i.e. if it makes sense in the domain being modeled).
5. For each interaction change (represented by an arrow), if there are different classes in the source interaction and the target interaction, define a relationship between these classes. It is also necessary to verify if this relationship is semantically correct (i.e. if it makes sense in the domain being modeled).
6. For each operation that appears in the user interaction diagrams, define an operation in the correspondent class.
7. In the end of the process, remove each class that remains without attributes.

We present here the definition of the conceptual schema synthesized from the user interaction diagrams of the case study. The notation used to represent the conceptual schema is UML [2].

Step 1: Definition of the classes. In each user interaction diagram, a class was defined for each element, set element and specific element. The following classes were identified for each use case:

CD (use cases 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)

Song (use cases 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13)

Artist (use cases 4, 7)

Type of music (use case 7)

Composer (use case 10)

Shopping Basket (use case 12)

Purchase (use case 2)

Step 2: Definition of the attributes.

A - The following attributes were identified from the information of the set elements of the user interaction diagrams:

CD: title, year, price, availability, image cover, country, copies, price-offer.

[2] Defined in [6] as "A functional dependency, denoted by X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation instance r of R. The constraint states that, for any two tuples t_1 and t_2 in r such that $t_1[X] = t_2[X]$, we must also have $t_1[Y] = t_2[Y]$. This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; or alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. We also say that there is a functional dependency from X to Y or that Y is functionally dependent on X."

[3] Defined in [6] como "A functional dependency X → Y in a relation schema R is a transitive dependency if there is a set of attributes Z that is not a subset of any key of R, and both X → Z and Z → Y hold.

Song: title, time, lyrics.

Artist: name.

Type of music: name.

Composer: name.

Shopping Basket: quantity.

Purchase: -

B - The following attributes were identified from the information entered by the users:

CD: title, year.

Song: title, sample.

Artist: name.

Type of music: -

Composer: name.

Shopping Basket: -

Purchase: payment-option, payment-plan, delivery-option, address, rcv-email, rcv-ads.

Step 3: Definition of the relationships.

A - The following relationships were identified from the attributes belonging to distinct classes but that appears as information of the same element:

Composer - Song

CD - Artist

CD - Shopping Basket

CD - Type of music

CD - Song

Song – Artist

B - The following relationships were identified from the different classes presented in the source and target interactions:

CD - Composer

CD – Song

CD – Artist

Artist - Type of music

Step 4: Definition of the operations. The following operations were identified from the user interaction diagrams:

CD: add to basket.

Song: listen to sample.

Artist: -

Kind of music: -

Composer: -

Shopping Basket: -

Purchase: -

Figure 5 shows the conceptual schema that results from these steps:

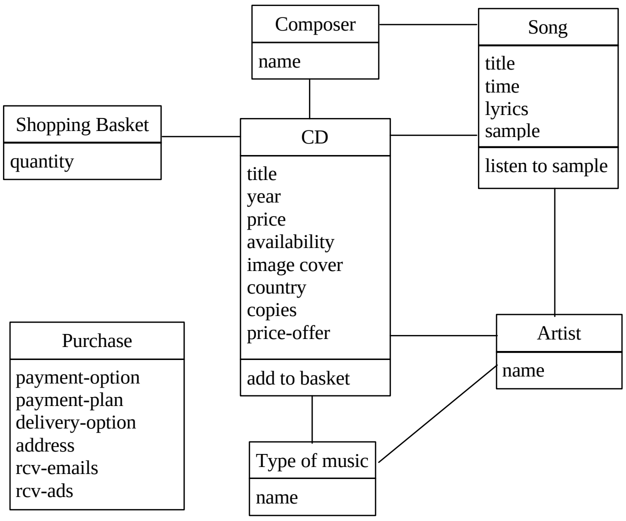


Figure 5 - Final resulting conceptual schema

TORWARDS NAVIGATIONAL DESIGN

Before explaining how some conceptual techniques can be extended to support the navigational design, we describe briefly the navigation design phase of OOHDM. As explained in the introduction, the requirements specifications, and user interaction diagrams will also be used to synthesize the navigation design of the application. Since a complete description of this process is outside the scope of this paper (see [9] for more details), we give here only a brief explanation, in order to allow the reader to have a high level understanding of the whole process.

OOHDM Overview

After conceptual modeling and before implementation, OOHDM suggests two phases: navigational design and abstract interface design [18, 22]. The navigational design phase yields navigational classes, links, anchors, contexts and InContext classes. The abstract interface design specifies how these navigational objects will be perceived and presented.

Navigational classes hold the information contents the user navigates through. These navigational classes are seen as a view of the underlying conceptual classes, since **a** navigational class may not have all the information content of a conceptual class, or may join the contents of more than one conceptual class. The links and anchors related to the navigational classes are also defined in the navigational design.

Depending on the task to be performed, the user can navigate through different sets of objects of the navigational classes (navigation contexts). For example, the user could navigate through the CDs belonging to Beatles or the Rock CDs published in the current year. Thus, the "next" or "previous" object of a set of objects that belong to a navigational class depends on the context the user is navigating. So, each navigation context has its own internal navigational structure, an entry point and associated indexes.

It may also occur that the objects have specific information contents according to their navigational context. For example, if a CD is accessed in the context of “Rock CDs”, it also shows its singer’s photo and short biography; if this same CD is accessed in the context of “CDs by Singer”, this information is not shown. For this reason, InContext classes are defined to represent the different information contents that objects can present when accessed within particular navigation contexts.

The abstract interface design specifies how the navigational objects will be perceived and presented, and it also specifies the interface objects representing the interactions. It is important to distinguish the navigation objects and interaction objects.

From Conceptual Design to Navigational Design

The scenarios, use cases, user interaction diagrams and conceptual schema defined in the conceptual design phase constitute the input for navigational design. Scenarios are important since they hold information about navigation. As previously said, good hypermedia (and therefore, web-based) applications involve judicious use of navigation facilities. This means that some interaction steps should be mapped onto navigational steps. The navigation design phase aims at designing the navigation topology of the application. Consequently, use cases and user interaction diagrams must be extended to include navigational aspects. For example, to represent an index, the following symbol is added to the user interaction diagram:

Object (data items)

To represent a navigational step, the following symbol is used (instead of a full arrow representing interaction):

- - - - - - →

In order to differentiate the original User Interaction Diagrams from those extended with information about navigation, we call the later User Navigation Diagrams. Figure 6 shows a user navigation diagram that is an extension of the user interaction diagram presented in figure 3. In this diagram, we assume that there are indexes for artists, CDs and songs. The following attributes are used to select an element of these indexes, respectively: artist name, CD title and song title. In the user navigation diagrams, the ellipsis in front of an element name means that the set elements can be accessed without returning to the index. So, when a CD is accessed, the other CDs can be directly accessed without returning to the index. But when a track or song is accessed, to go to the next track it is necessary to return to the index. The textual description of the use case must also be modified to incorporate these navigational aspects.

User navigation diagrams are also validated with the help of the user. In the validation process, the designer asks the users about navigational aspects, for example, if a navigational context has an associated index and its internal navigation. The validation process is similar to that presented in the section 3.4. After the designer has validated the user navigation diagrams with the users, context schemas representing the navigational contexts and their access structures [22] are defined based in these user navigation diagrams.

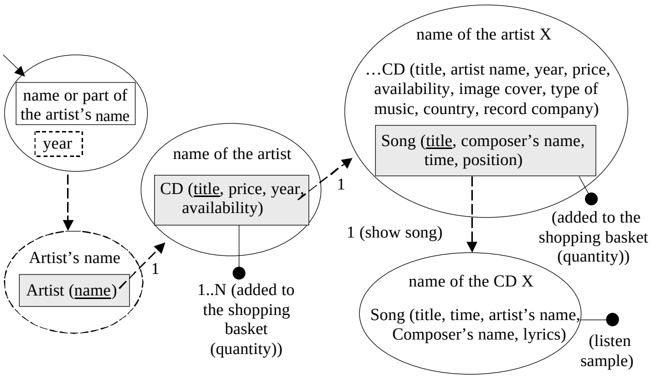

name or part of the artist’s name
year
Artist’s name
Artist (name)
1
name of the artist
CD (title, price, year, availability)
1..N (added to the shopping basket (quantity))
1
name of the artist X
…CD (title, artist name, year, price, availability, image cover, type of music, country, record company)
Song (title, composer’s name, time, position)
(added to the shopping basket (quantity))
1 (show song)
name of the CD X
Song (title, time, artist’s name, Composer’s name, lyrics)
(listen sample)

Figure 6 – User navigation diagram

Figure 7 presents the context schema based on to the user navigation diagram shown in figure 6. Context schemas are complemented by context cards, that have additional information about the contexts, such as the internal navigational structure (sequential, circular, via index).

User Navigation Diagrams are also used to obtain the navigational schema due to the fact that, in the navigation design phase, some data items that do not appear as attributes of a specific conceptual class can become attributes of its corresponding navigation class. For example, although the singer name is an attribute only of the class *Artist*, it could also become an attribute of the navigation class CD during navigational design. This can be verified in the user navigation diagram, where the singer’s name appears as one of the data items presented with CD.

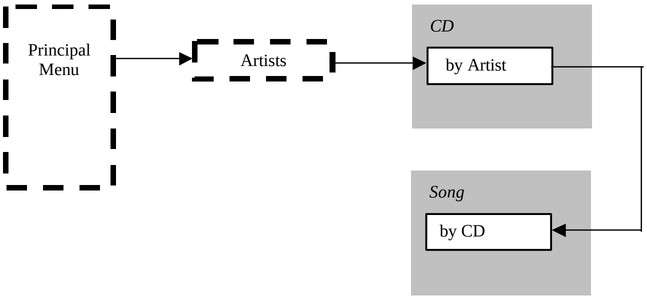


Figure 7 - Context schema

RELATED WORK

There are several approaches of requirements gathering using scenarios and use cases. In general, these approaches make use of scenarios and use cases in a different way of what we are proposing. We are not aware of any similar technique to user interaction diagrams for graphically representing the interaction specified in the use cases.

[7] presents an approach of HCI design oriented to web sites which is based on scenarios. Since this approach is applied to the redesign of an existing web site, the scenarios also describe interface and navigation aspects.

In [23], use cases are used to elicit the application requirements from the users. Scenarios are automatically generated from these use cases and are used to refine and validate the requirements.

[12] proposes an approach based on user stories and use cases for gathering the requirements. A user story can be seen as a simpler scenario, where an agent performs a physical action on an object. Thus, user stories contain, at least, an agent, an action and an object, and also include the intention of users. Use cases are specified from the user stories. The authors also rely on some experientialist concepts (stories, mental spaces, projection and blends) to aid the mapping between user stories and use cases. They propose a link among use cases and user stories to facilitate the access to the intentionality and motives. Once more, however, in both user stories and use cases, the interactions between the users and the system are specified as a textual description, i.e., there is no diagrammatic notation to represent the interaction nor guidelines to define the conceptual schema from the use cases as we do in our work.

In the Unified Process proposed in [15], the requirements are also defined by use cases. However, scenarios are not used to specify those use cases. In the initial phase, i.e. requirements gathering, user interface prototypes help to understand and specify the requirements. The authors do not propose any diagrams that focus specifically on the interaction between the user and application. The authors suggest that use cases can also be described by statechart diagrams, activity diagrams, collaborations, and sequence diagrams. We believe these diagrams rely on early decisions such as definition of the classes and attributes that shall have to be validated later in the process.

Interaction and collaboration diagrams [13] require the previous definition of the classes and their operations. Although interaction and collaboration diagrams are simpler than our user interaction diagrams, it is not easy to infer the navigation and user interface from these diagrams.

Activity diagrams [2] and Statechart diagrams [10], in turn, do not present all the information regarding the interaction between the user and the application, neither the user is referenced by the diagram.

Another known approach for requirement analysis is the use of DFDs as suggested by Modern Structured Analysis Techniques [24]. Although DFDs allow for the representation of the user as an external entity, all data repositories are defined against a co-existent conceptual model (e.g. entity-relationship diagram). Thus, such conceptual model and the corresponding DFDs will have to be validated later during the process. But, we agree that DFDs are useful to explicit the interactions (i.e. input and output) between process, although DFDs do not evidence the interactions with the user and can neither represent navigational aspects.

CONCLUSION

We have presented a method for requirements gathering, interaction specification and conceptual design of web applications. The basis of this method are scenarios, use cases and user interaction diagrams, a graphical notation proposed here. User interaction diagrams represent the interaction between the user and the application, and are used to validate use cases and constitute the basis for the definition of the conceptual schema. As shown in this paper, user interaction diagrams can also be extended to support navigation aspects.

User interaction diagrams were proposed to be used in a higher abstraction level, without be concerned on user interface and design details. When we specify a user interaction diagram we do not worry about the difference between navigation interaction and interface interaction. This difference will be treated later during the navigational design of a web application, when user navigation diagrams are specified. An accurate definition for the navigation interaction concept is crucial for a good navigational design, which in turn is crucial for a good web application design.

We have also proposed guidelines to define the conceptual schema from the user interaction diagrams. Some of these guidelines are based on the very know concept of functional dependency. The advantage in verifying these diagrams against functional dependencies to obtain a conceptual schema is that the information content is already validated. If we applied functional dependency check to the scenarios, the conceptual schema would have to be refined after validating use cases and user interaction diagrams.

The introduction of user interaction diagrams in the conceptual design stage and the corresponding user navigation diagrams in the navigational design stage is the key for the success of web application development and we believe that our method helps the designers to achieve this goal.

The proposed method was experimented only with OOHDM. But since it deals with the first phases of the development cycle, it can also be used with the other existing methods for hypermedia applications design (EORM, RMM, HDM).

We are also applying this method to a project with the Training Division of Petrobras, a Brazilian oil company. The site of this division is being redesigned. 26 users belonging to 6 types of actors have been interviewed and asked to describe scenarios. 122 scenarios were described and 52 use cases were specified. At the moment, we are about to define the conceptual schema based on the diagrams that represent these use cases.

ACKNOWLEDGMENTS

Patrícia Vilain would like to thank CAPES for sponsoring this work. Daniel Schwabe and Clarisse Sieckenius de Souza would like to thank CNPq for supporting their research. All authors would like to thank the participants of the case study.

REFERENCES

1. Booch, G. Object-Oriented Analysis and Design with Applications - 2nd Edition, Benjamin/Cummings Publishing Company, 1994.
2. Booch, G., Jacobson, I., and Rumbaugh, J. The Unified Modeling Language User Guide, Addison - Wesley, 1999.
3. Carroll, J.M. Scenario-Based Design: Envisioning Work and Technology in System Development, John Wiley & Sons, 1995.
4. Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P. Object-Oriented Development: The Fusion Method, Prentice Hall, 1993.
5. Eco, U. , and Sebeok, T.A. The Sign of Three: Dupin, Holmes, Peirce. Indiana University Press, 1984.
6. Elmasri, R., and Navathe, S.B. Fundamentals of Database Systems, Second Edition, Benjamin/Cummings, 1994.
7. Erskine, L.E., Carter-Tod, D.R.N., and Burton, J.K. Dialogical techniques for the design of web sites. *Int. Journal Human-Computer Studies, 47* (1997), 169-195.
8. Garzotto, F., Paolini, P., and Schwabe, D. HDM - A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems 11*, 1 (January 1993), 1-26.
9. Guell, N. A description on the use of User Centered Scenarios to synthesize Navigation design of hypermedia applications (in Portuguese). Tech. Report MCC 10/98, Dept. of Informatitcs, PUC-Rio 1998. 18p. Available in ftp://ftp.inf.puc-rio.br/pub/docs/techreports/ 98_10_schwabe.pdf.gz.
10. Harel, D. Statecharts: a visual formalism for complex systems. *Science of Computer Programming 8* (1987), 231-274.
11. Isakowitz, T., Stohr, E., and Balasubramanian, P. RMM: A methodology for structuring hypermedia design. *Commun. ACM 38*, 8 (August 1995), 34-44.
12. Imaz, M., and Benyon, D. How Stories Capture Interactions, in *Proceedings of Human-Computer Interaction - INTERACT'99*, IOS Press, 321-328.
13. Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. Object-Oriented Software Engineering - A Use Case Driven Approach, Addison-Wesley, 1992.
14. Jacobson, I. The Use-Case Construct in Object-Oriented Software Engineering. Scenario-Based Design: Envisioning Work and Technology in System Development. John Wiley & Sons, 309-336, 1995.
15. Jacobson, I., Booch, G., and Rumbaugh, J. The Unified Software Development Process, Addison-Wesley, 1999.
16. Lange, D. An Object-Oriented Design Method for Hypermedia Information Systems (1994), 366-375.
17. Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. Human-Computer Interaction, Addison-Wesley, 1994.
18. Rossi, G., Schwabe, D., and Lyardet, F. Web Application Models Are More than Conceptual Models, in *Proceedings of ER'99* (Paris, France, November 1999), Springer, 239-252.
19. Rossi, G., Schwabe, D., and Lyardet, F. Improving Web information systems with navigational patterns, in *Proceedings of 8th International World Wide Web Conference* (Toronto, Canada, May 1999), Elsevier Science, 589-600.
20. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. Object-oriented modeling and design. Englewood Cliffs, 1991.
21. Rumbaugh, J. Getting Started: Using Use Cases to Capture Requirements. *Journal Object-Oriented Programming* (September 1994), 8-12.
22. Schwabe, D., and Rossi, G. An object-oriented approach to Web-based application design. *Theory and Practice of Object Systems (TAPOS)* (October 1998), 207-225.
23. Sutcliffe, A.G., Maiden, N.A.M., Minocha, S., and Manuel, D. Supporting Scenario-Based Requirements Engineering. *IEEE Transactions on Software Engineering 24*, 12 (December 1998), 1072-1088.
24. Yourdon, E. Modern Structured Analysis. Englewood Cliffs, 1989.