Using Task Descriptions for the Specification of Web Application Requirements[1]

Pedro Valderas, Joan Fons and Vicente Pelechano

Department of Information System and Computation.
46022, Technical University of Valencia, Spain
{pvalderas, jjfons, pele}@dsic.upv.es

Abstract. There are a significant number of proposals for modelling and developing Web Applications, but very few of them state rigorously how to elicit and represent requirements, and how to go from the requirements specification to the conceptual schema with a sound methodological basis. This work presents an approach to capture Web application requirements by means of: (1) the identification of the tasks that users must be able to achieve and (2) the description of these tasks from the point of view of the interaction that the user requires of the web application. In addition, we show how the navigational structure of a Web application can be systematically derived from the task-based requirement specification.

1 Introduction

There are a significant number of proposals (OOHDM [2], UWE [3], WSDM [4], OOH [5], OOWS [6], etc.) that provide a methodological solution for developing Web Applications. However, these proposals are mainly focus on defining Web applications from conceptual schemas that allow them to systematically obtain implementations. Very few of them state rigorously with how to elicit and represent requirements, and how to go from the requirements specification to the conceptual schema with a sound methodological basis.

In the context of requirement engineering, traditional methods such us Constantine et al. [10], Jaaksi [11], Leite et al. [12], Rosenberg et al. [13] have done an acceptable work in specifying structure and behaviour requirements. But a Web Application requires considering some other particular aspects that are not properly addressed by traditional requirement engineering methods. In concrete, navigation becomes first-order citizen, and the requirement specification step must consider it accordingly.

In this work, we present an approach to capture Web application requirements. This approach is based on the *task* metaphor, which is widely

[1] The work reported in this paper has been funded by the MEC under grant TIN2004-03534 and cofinanced by FEDER

accepted for the capture of functional requirements; however it is reoriented to capture, in a adequate way, the navigational structure that fits the user's needs. We extend the traditional task descriptions used for the functional requirement specification by introducing information about the *interaction between the user and the system*. In addition, we present a methodological guide that allows us to systematically obtain the navigational structure of a Web application from tasks descriptions. This, allows us to better understand the capabilities of our approach to capture navigation at the requirement level.

This paper is organized as follows: Section 2 presents a task-based method for the specification of Web application requirements. Section 3 explains how the navigational structure of a Web application is systematically derived from task descriptions. Finally, conclusions and future work are presented in section 4. To clearly introduce our work, an E-commerce application like the Amazon web site (thereafter known as the "Amazon example") has been taken as a case study.

2 A Task-Based Method for the Specification of Web application Requirements

In this section, we present a method to specify Web application requirements from the description of the tasks that users must be able to achieve. This method is divided into two main stages:

(1) Identification of the tasks that describe the user's needs. To make this job easy, we propose the definition of a task taxonomy.
(2) Description of the set of tasks. In this case, we propose a technique based on both UML activity diagrams and information templates.

2.1 Task Identification

In order to easily identify tasks, we propose the construction of a task taxonomy taking a *statement of purpose*, which describes the goal for which the application is being built, as the starting point. The statement of purpose is considered as the most general task. From this task, a progressive refinement is performed and more specific tasks are obtained from more general ones. Tasks are decomposed into subtasks by following structural or temporal refinements. The *Structural* refinement (represented by solid lines, see Figure 1) decompose complex tasks into simpler subtasks. The *Temporal* refinement (represented by dashed lines, see Figure 1) provide constraints for ordering tasks that are all children of a single task according to the task logic. To define these temporal constraints we propose the use of the temporal relationships introduced by the CTT approach (ConcurTaskTree) [7]. Due to space constraints we only present the three relationships that are used in the case study introduced in this paper:

– T1 []>> T2, *Enabling with information passing*: when T1 is terminated then T2 is activated. In addition, when T1 task terminates it provides some value for task T2 besides activating it.
– T1 |> T2, *Suspend-Resume:* T1 can be interrupted by T2. When T2 terminates then T1 can be resumed.
– T1*, *iteration*: the task can be achieved several times.

As we have said above, the task taxonomy is used to identify the tasks that describe the user's needs. The task taxonomy is not used to describe tasks. Tasks are described by means of activity diagrams [1] because they allow us to better capture the navigational structure of a Web application (explained below in detail). In this sense, the task taxonomy is finished when elementary tasks are obtained. An *elementary task* is defined as a task that when divided into subtasks, atomic actions are obtained. Figure 1 shows the task taxonomy that we obtain from the statement of purpose of the Amazon example. In order to easily identify the elementary tasks they are circled with a thicker line.

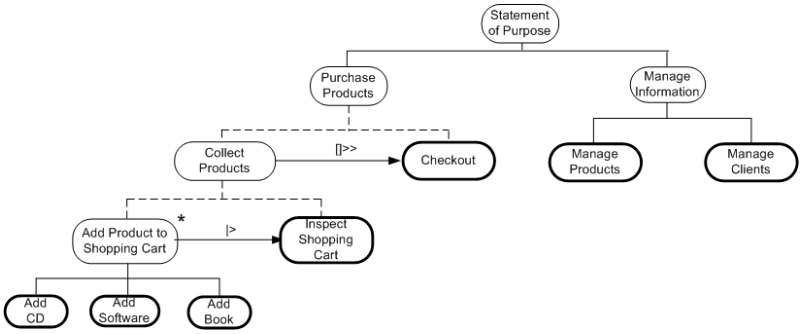

Fig. 1. The Task Taxonomy of the Amazon example

We briefly describe the task taxonomy of Figure 1:
– The statement of purpose is decomposed by means of a structural refinement (solid line) into two tasks: *Purchase Products* and *Manage Information*.
– The task *Purchase Products* is decomposed by means of a temporal refinement (dashed line) into *Collect Products* and *Checkout*. The relation between them is *enabling with information exchange*. Indeed, first products should be collected into de shopping cart before checkout is possible. The information that needs to be exchange is the shopping cart.
– *Collect Products* is decomposed into *Add Product to Shopping Cart* (which can be repeated) and *Inspect Shopping Cart*. The relation between both tasks is *suspend-resume*, which indicates that *Add Product to Shopping Cart* can be interrupted by *Inspect Shopping Cart* at any point. It will be reactivated from the state reached before the interruption once *Inspect Shopping Cart task* is ended.

– *Add Product to Shopping Cart* product task is decomposed by means of a structural refinement into the tasks *Add CD*, *Add Software* and *Add Book*.
– The task *Manage Information* is decomposed by means of a structural refinement into *Manage Products* and *Manage Clients*.

Tasks inherit the temporal constraints of the ancestors. For instance, in Figure 1 *Add Book* is a sub-task of *Add Product to Shopping Cart* and since *Add Product to Shopping Cart* can be suspended by *Inspect Shopping Cart*, this constraint will also apply to *Add CD*.

Once elementary tasks are identified we must describe how they should be achieved. Next, we introduce a strategy to do this.

2.2 Description of tasks

In the traditional specification of functional requirements a task is described from the set of actions that the system and the user perform to obtain a certain result. In order to better capture the navigational properties of a Web application we extend these descriptions by introducing information about the *system-user interaction*, indicating explicitly when (at which exact moment) it is performed. To do this, we introduce the concept of **interaction point (IP).** Two kinds of interactions can be performed in an IP:

(1) *Output Interaction*: the system provides the user with information and/or access to operations which are related to an entity[2]. The user can perform several actions with both the information and the operations: the user can select information (as a result the system provides the user with new information) or the user can activate an operation (as a result the system carries out an action).

(2) *Input Interaction*: the system requests the user to introduce information of an entity. The system uses this information to correctly perform a specific action (for instance, the client information needed to carry out an on-line purchase). In this case, the only action that user can perform is the information introduction.

In this way, a task is described as a process where the system carries out several *actions* sometimes delaying them in order to *interact* with the *user* by means of an IP. As far as the system actions, two kinds are proposed: *(1) Functionality Execution* that are actions that change the system state and (2) *Information Search* that are actions that only query the system state.

In order to perform descriptions based on IPs we propose the use of UML *activity diagrams* [1] where:

– Each node (activity) represents an IP (solid line) or a system action (dashed line). IPs are stereotyped with the *Output* or the *Input* keyword to indicate the interaction type. System actions are stereotyped with the *Function* or the *Search* keywords to indicate their types.

[2] Any object of the real world that belongs to the system domain (e.g. client, product, invoice, etc)

– In the *Output IPs* the number of information instances[3] that the IP includes (cardinality) is moreover depicted as a small circle in the top right side of the primitive.
– As far as the *Input IPs*, we have said that they are used by the system to correctly perform a specific action. To capture that this kind of IPs exclusively depends on a system action and it does not take part in the general process of the task, nodes that represent both elements (input IP and system action) are encapsulated into dashed squares.
– Finally, each arc represents (1) a user action if the arc source is an IP or (2) a node sequence if the arc source is a system action. If an arc represent a user action (the arc source is an IP), it can be either an activation of an operation, if the arc target is a system action, or an information selection, if the arc target is another IP.

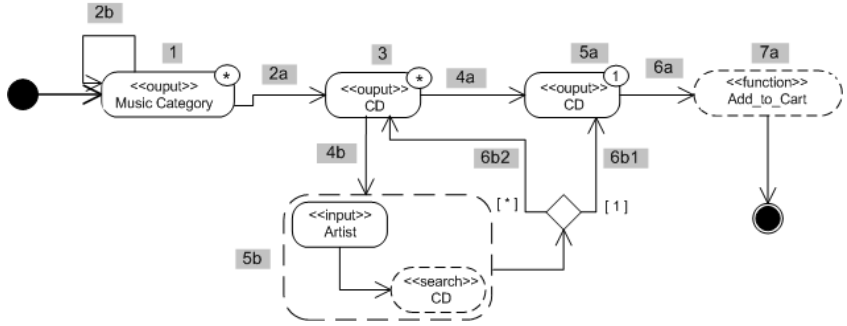

Fig. 2. *Add CD* Elementary Task.

Continuing with the Amazon example, the *Add CD* elementary task is described in Figure 2 (the shaded numbers are not part of the notation). This task starts with an *Output* IP where the system provides the user with a list (cardinality *) of music categories (1). From this list, the user can select a category (2a and 2b). If the category has subcategories the system provides again the user with a list of (sub) categories (2b). If the selected category has not subcategories (2a) the system informs about the CDs of the selected category by means of an *Output* IP (3). From this IP the user can perform two actions: **A)** select a CD (4a) and then the system provides the user with a description of the selected CD (5a). **B)** Activate a search operation (4b) and then the system performs a system action which searches the CDs of an artist (5b). To do this, the user must introduce an artist by means of an *Input* IP. If the search returns only one CD, the system provides the user with its detailed description (6b1). Otherwise, the system provides the user with a set of CDs

[3] Given a system entity (e.g. client), an information instance is considered to be the set of data related to each element of this entity (Name: Joseph, Surname: Elmer, Telephone Number: 9658789).

(6b2). Finally, when the user has obtained a CD description (5a) he/she can activate the *Add_to_Cart* operation (6a) and then the system performs an action which adds the selected CD to the shopping cart (7a).

IP-based descriptions are proposed to capture the navigational properties of Web applications. In order to make this capture easy, details about the information exchanged between the user and the system are not described (we just indicate the entity which the information is related to). In the same way, details about how the system achieves each action are neither described. This information is specified in later steps. This, allows us to provide a high level of independency among different kinds of requirements.

In order to describe the system actions, we propose a strategy based on sequence diagrams that is explained in [27]. Details about the information exchange are described using an information template technique that is next introduced.

Describing the system data. The information that must be stored in the system is defined by means of a template technique that is based on data techniques such as [8] or the CRC Card [9]. We propose the definition of an information template (see Figure 2) for each entity identified in the description of a task. In each template we indicate an identifier, the entity and a *specific data* section. In this section, we describe the information in detail by means of a list of specific features associated to the entity. For each feature we provide a name, a description and a data type. In addition, we use these templates to indicate the information shown in each IP. For each feature we indicate the IP/s where it is shown (if there is any). To identify an IP we use the next notation: *Output (Entity, Cardinality)* for Output IPs and *Input (Entity, System Action)* for Inputs IPs.

Identifier:	O1			
Entity:	CD			
Specific Data:	*Name*	*Description*	*Type*	*IPs*
	Title	Title of the CD	String	Output(CD,*), Output(CD,1)
	Year	Recording year of the CD	String	Output(CD,1)
	Artist name	Artist that has recorded the CD	String	Output(CD,*), Output(CD,1)
	Songs	Songs list of the CD	String[]	Output(CD,1)
	Comments	A brief commentary of the CD	Text	Output(CD,1)
	Frontal Cover	CD cover frontal	Image	Output(CD,*), Output(CD,1)
	Price	Price of the CD	Number	Output(CD,*), Output(CD,1)
	Purchase times	Times that the CD has been purchased	Number	
	Client Profiles	Profiles of the clients that have purchased the CD	String[]	

Fig. 3. Information template.

According to the template in Figure 3, the information that the system must store about a CD is (see the specific data section): the CD title, the artist name, the front cover and the price which are shown in the IPs Output(CD,1) and Output(CD,*) (IPs defined in the *Add CD* elementary task, see Figure 3); the recording year, some comments about the CD and the list of songs which

are only shown in the IP Output(CD,1); and finally, times that a CD has been bought and the client profiles that usually purchase it which are not shown in any IP.

3 Extracting the Navigational Structure of a Web application from Task Descriptions

In this section we present some guidelines to extract the navigational structure of a Web application from the task-based requirement specification presented above. To represent a navigational structure we use the abstractions proposed by the OOWS method [6]. In this sense, a (necessary) brief overview of the OOWS navigational model is next presented.

3.1 The OOWS navigational model: an Overview

The OOWS navigational model is made up of a set of navigational maps that describe the navigation allowed for each kind of user. A navigational map is represented by a directed graph (which defines the navigational structure) whose nodes are *navigational contexts* and its arcs denote *navigational links*. Figure 4A shows the visitor navigational map of the Amazon example. On one hand, a navigational context (represented by an UML package stereotyped with the *«context»* keyword) defines a view over the classes of the class diagram that allows us to specify the information that is shown in the context (class attributes) and the operations that the user can activate (class operations). The navigational context CD (see Figure 4B) provides the user with information about CDs (title, year, songs, comments, cover and price) and about their artists (name). Moreover, the *Add_to_Cart* operation can be activate by the user. On the other hand, a navigational link represents navigational context reachability: the user can access to a navigational context from a different one if a navigational link between both has been defined.

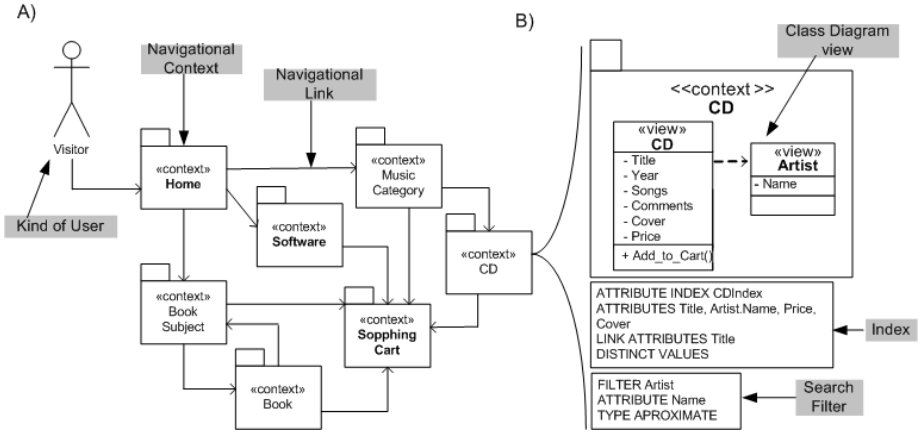

Fig. 4. The OOWS Navigational Model

In addition, for each context, we can also define: (1) *Search filters*, which are mechanisms that allow us to filter the space of objects that retrieve the navigational context. The CD navigational context allows the user to find all the CDs of a specific artist (see search filter in Figure 4B) and (2) *Indexes*, which are structures that provide an indexed access to the population of objects. Indexes create a list of summarized information allowing the user to choose one item (instance) from the list. This selection causes this instance to become active in the navigational context. The navigational context of Figure 4B provide the user with a list of summarized information where for each CD, the title, the artist name, the price and the cover are shown.

3.2 From Task Descriptions to OOWS Navigational Models

In this section, we explain how the navigational structure of a Web application (described by means of the OOWS abstractions) is systemically derived from task descriptions. First, we explain how navigational contexts are detected and defined. Next, we explain how detect navigational links. Finally, detection of indexes and search filters are also presented.

Detecting Navigational contexts. We detect navigational contexts from the Outputs IPs defined in the tasks descriptions. An Output IP represents a step of a task where the system provides the user with some information about an entity. In the OOWS navigational model, information is provided to the user by means of navigational contexts. Then, each Output IP derives into a navigational context except for those IPs than both inform about multiple instances (cardinality *) of one entity and allow the user to access to another IP which informs about only one instance (cardinality 1) of the same entity. These situations are explained below. On the other hand, the view of each navigational context is derived from: (1) the information template features that are shown in the IP (which define the class attributes) and the *function* system actions that can be activate from the IP (which define the class operations).

Figure 5 shows the navigational contexts detected from the task *Add CD*. The Output(Music Category,*) IP derives into the navigational context *Music category*. The Output(CD,1) IP derives into the navigational context *CD*. Any context is derived from the Output(CD,*) IP because it allow the user to access to Output(CD,1) IP (same entity, only one instance). In addition, Figure 5 also shows how the navigational context CD is defined. On one hand, the features specified in the CD entity template (which are shown in the IP Output(CD,1)) define the class attributes. Furthermore, the *Add_to_Cart* operation is defined in the CD class because the *Add_to_Cart* system action can be activated from the Output(CD,1) IP.

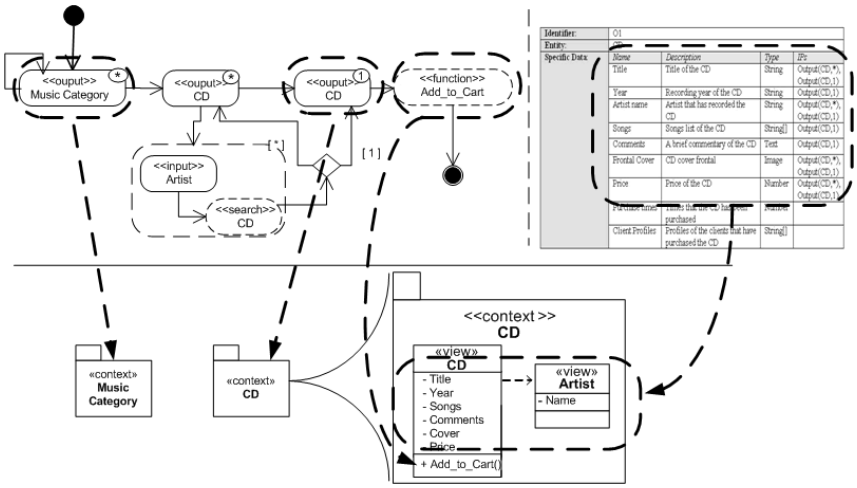

Fig. 5. Context definition form task descriptions

Detecting Navigational Links. Navigational links are detected from the activity diagrams that describe each task. We define a navigational link between two navigational contexts if the IPs which the contexts have been detected from are: (1) connected by means of an arc or (2) connected through an IP which has not derived into any context. In addition, the temporal relationships defined in the task taxonomy also allow us to identify navigational links. For instance, if a *suspend/resume* relationship has been defined between two task T1 and T2, navigational contexts derived from T2 must be accessible from navigational contexts derived from T1. Then, navigational links among T1 navigational contexts and T2 navigational contexts are defined.

Figure 6 shows the navigational links defined from the task *Add CD*. On one hand, a navigational link is defined between the contexts *Music Category* and *CD* because the Output(Subject,*) IP and the Output(Book,1) IP (IPs which contexts are detected from) are connected through the Output(Book,*) IP (which has not derived into any context). On the other hand, two navigational links are defined to connect the contexts *Music Category* and *CD* to the context *Shopping Cart* (which has been derived from the *Inspect Shopping Cart* task) due to a *suspend/resume* relationship. Taking into account that a task inherits the temporal relationships of its parent tasks, the *Add CD* elementary task is connected to the *Inspect Shopping Cart* elementary task by means of a Suspend/Resume relationship (inherited from the Add *Products to Shopping Cart* task). Then, the navigational contexts derived from the task *Add CD* are linked to the navigational context derived from the task *Inspect Shopping Cart*.

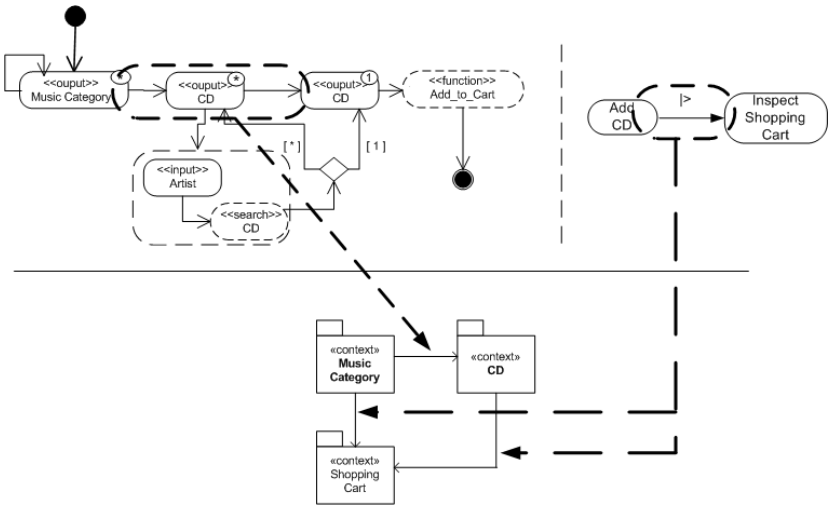

Fig 6. Identification of the navigational links.

Detecting Indexes and Search Filters. On one hand, indexes are detected from those output IPs that both informs about multiple instances of one entity (cardinality *) and provides the user with access to a second IP that informs about only one instance of the same entity (cardinality 1). These IPs are defined to allow the user to compare a list of elements (IP instances) among themselves in order to select the desired one. Then, these IPs define an index in the navigational context detected from the second IP. Index attributes are detected from the template features that are shown in the first IP. On the other hand, search filters are detected from *search* system actions. Each *search* system action that is activated from an output IP which has defined either a navigational context or an index of a navigational context maps to a search filter of the navigational context. Filter attributes are defined from the template features that are request in the input IP which allow the user to introduce the search criterion.

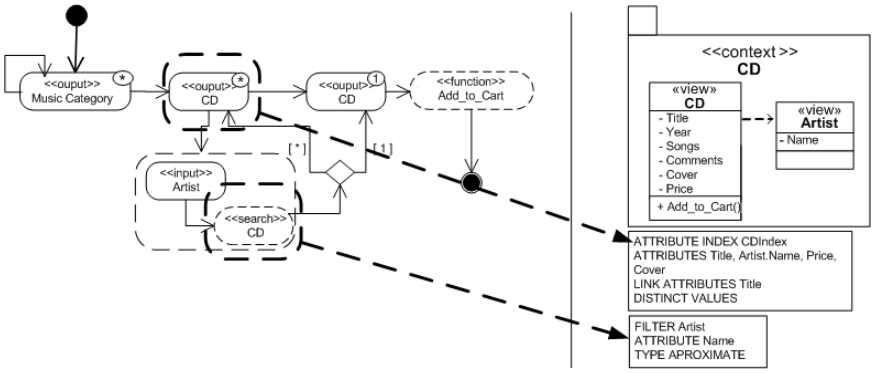

Fig. 7. Identification of indexes and search filters.

Figure 7 shows the information access mechanisms of the *CD* navigational context. On one hand, the Output(CD,*) IP defines an index whose attributes are detected from the template features that are shown in the IP. On the other hand, a search filter is defined into the *CD* navigational context because a search system action can be activated from the IP that generates an index of this context, (Output(Book,*)). Attributes of the filter are obtained from the template features that are request in the input IP that allows the user to introduce the search criterion (an artist).

Implementation Issues. Figure 8 shows the implementation of the contexts derived from the task *Add CD*. Figure 8A shows the implementation of the context *Music Category*. Figures 8B and 8C implement the context *CD* (index of CDs and CD description). Furthermore, we can see how the implemented Web pages provide support to achieve the task *Add CD* according to its description (see Figure 2). This allows us to check that navigation has been correctly captured in the task-based requirement specification.

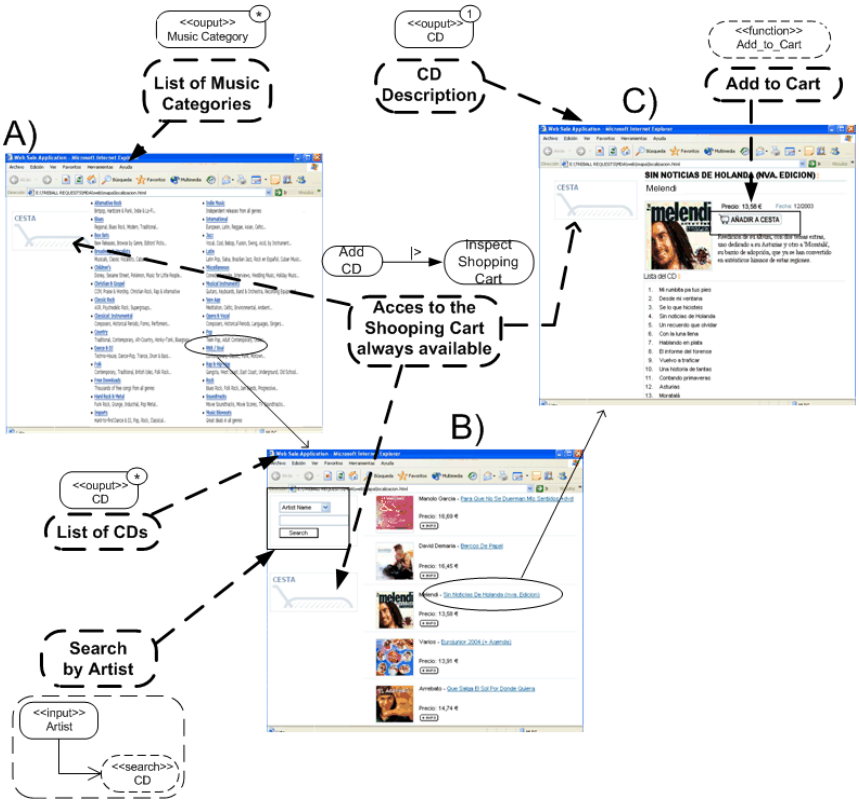

Fig. 8. Implementation of the contexts *Music Category* and *CD*

4 Conclusions

In this work we have presented an approach to capture the requirements of Web application by (1) identifying the tasks that users must achieve and (2) describing these tasks from the interaction that the user requires of the web application. In addition, we have shown how the navigational structure of a Web application can be systematically derived from a task description.

As a proof of concept this proposal has been put into practice successfully in the development of small and medium-size web applications, including the DSIC Department Web Site (http://www.dsic.upv.es), the OOMethod Group Web Site (http://oomethod.dsic.upv.es) and the Web application of the Development Cooperation Center (http://www.upv.es/ccd). In all these cases, the requirements specification and its corresponding Web conceptual schema were obtained according with the introduced approach.

Finally, we are currently defining a wizard that asks the user by means of a guided process in order to systematically detect and describe tasks. This wizard will allow us to hide the possible complexity of our notation making the definition of complex Web applications easier.

References

[1] Object Management Group. Unified Modeling Language (UML) Specification Version 2.0 Final Adopted Specification. www.omg.org, 2003.

[2] D. Schwabe, G. Rossi, and S. Barbosa. Systematic Hypermedia Design with OOHDM. In ACM Conference on Hypertext, Washington, USA, 1996.

[3] N. Koch. Software Engineering for Adaptive Hypermedia Applications. PhD thesis, Ludwig-Maximilians-University, Munich, Germany, 2000.

[4] O. De Troyer and S. Casteleyn. Modelling Complex Processes fro Web applications using WSDM. In International Workshop on Web Oriented Software Technologies. Oviedo, Spain. 2003 pp 1,12.

[5] J. Gómez, C. Cachero, O. Pastor. Extending an Object-Oriented Conceptual Modelling Approach to Web Application Design. June 2000. CAiSE'2000, LNCS 1789, Pags 79-93.

[6] J.Fons, V. Pelechano, M. Albert, and O. Pastor. Development of Web Applications from Web Enhanced Conceptual Schemas. In ER'03, volume 2813 of LNCS. Springer, 2003.

[7] F. Paternò, C. Mancini and S. Meniconi, 1997. "ConcurTaskTrees: a Diagrammatic Notation for Specifying Task Models", INTERACT'97, Chapman & Hall, 362-369.

[8] A. Durán, B. Bernárdez, A. Ruiz and M. Toro. A Requirements Elicitation Approach Based in Templates and Patterns. WER'99, Buenos Aires (Argentina), 1999. pp. 17-29.

[9] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object–Oriented Software*. Prentice–Hall, 1990.

[10] L.L Constantine., L.A.D. Lockwood "Software for Use: A practical Guide to the Models and Methods of Usage-Centered Design". Addison Wesley 1999.

[11] A. Jaaksi. *Our Cases with Use Cases*. JOOP Vol 10, Nº9, Febraury 1998, pp 58-65.

[12] J.C. Leite, G. Rossi, F. Balaguer, V. Maiorana, G. Kaplan, G. Hadad, and A. Oliveros, *Enhancing a Requirements Baseline with Scenarios.*, RE'97, Anapolis. IEEE, 1997, 44-53.

[13] D. Rosenberg, K. Scott, "Use Case Driven Object Modelling with UML". Addison Wesley, 1999.