From Crosscutting Concerns to Web Systems Models

Pedro Valderas[1], Vicente Pelechano[1], Gustavo Rossi[2], and Silvia Gordillo[2]

[1] Department of Information Systems and Computation
Technical University of Valencia, Spain
{pvalderas, pele}@dsic.upv.es
[2] LIFIA, Facultad de Informática, UNLP, La Plata
{gustavo, gordillo}@lifia.info.unlp.edu.ar

Abstract. In this paper we present a novel approach for dealing with crosscuting concerns in Web applications from requirements to design. Our approach allows to clearly decoupling requirements that belong to different concerns; these concerns are separately modeled and specified by using the task-based notation proposed by OOWS Web Engineering approach to specify requirements; we next show how we integrate task descriptions corresponding to different concerns to obtain a unified requirements model that is the source of a model-to-model and model-to-code generation process that allows us to obtain fully operative web application prototypes that are built from tasks descriptions.

1 Introduction

Even simple Web applications must deal with a myriad of concerns (functional and non-functional), each one of them encompassing multiple requirements. Some of these concerns crosscut and consequently the corresponding software artifacts may be tangled. This problem has been addressed in the field of Aspect-oriented software development (AOSD) [7,13] which encapsulate crosscutting concerns in separate modules, known as *aspects*, and composition mechanisms are later used to weave them back with other core modules, at loading time, compilation time, or run-time.

In our research we are interested however in the fact that crosscutting concerns are present well before the implementation, such as in requirements engineering as shown in [3]. Separating concerns from requirements allows modularizing those concerns that can not be easily specified as a single use case or task. Composition, on the other hand, apart from allowing the developers to picture the whole system, allows them to identify conflicting situations whenever a concern contributes negatively to others. Unfortunately so far mature Web engineering methods such as [5,14,8] do not offer primitives and composition mechanisms for advanced separation of concerns.

In [9] we presented an approach for identifying and composing navigational concerns in Web applications using concepts borrowed from aspect-oriented software. Using our approach we can detect cross-cuttings among concerns early in the development process and assess the impact of the crosscutting in the corresponding design models. Though our research was performed in the context of the OOHDM [12] design framework and uses UIDs [10] to describe application requirements, the ideas can be applied with other design methods as is the OOWS method. The OOWS

B. Benatallah et al. (Eds.): WISE 2007, LNCS 4831, pp. 573–582, 2007.

method introduces a notation to capture Web application requirements which is based in the task metaphor. Task descriptions are enriched with information about the interaction between the user and the system. It also includes a method to derive a OOWS navigational model automatically from these task based descriptions. In this way, the OOWS models can be further transformed into fully operative Web applications therefore allowing to completely bridging the gap between requirements elicitation and web application development.

In this paper we present a result of our research work combining techniques for separation of concerns in requirements engineering [16], with approaches for automatic derivation of Web design models from requirement models. The main contributions of the paper are the following:

- We present a novel approach to model requirements of Web applications as task-based representations of different concerns.
- We show how to obtain an integrated representation that can be later transformed into a unified design model.

Section 2 introduces the re-interpretation that we have done of the task-based method for Web application requirements specification in order to be used as technique for the description of concerns. Furthermore, an overview of the strategy proposed to automatically obtain Web application prototypes from task descriptions is also presented. Section 3 explains how the techniques for the integration of concerns are applied in the tasks-based method. Section 4 explains the different strategies that can be followed to support the integration of concerns in the generation of Web application prototypes. Finally, conclusions and further work are presented in Section 5.

2 Background: The Task Based Approach of OOWS

Section 2.1 introduces a technique for describing concerns by using the OOWS task-based notation for specifying requirements. Section 2.2 presents an overview of the strategy that allows us to obtain Web application prototypes from task descriptions.

2.1 Using Tasks for Describing Concerns

Following [1] we consider a concern to be a cohesive set of requirements which deal with the same application theme. Considering the interactive nature of Web applications, we model each concern by modeling the underlying tasks of the concern's requirements. We focus particularly on those functional concerns which involve interaction and navigation (called navigational concerns in [9]), because they define the skeleton of a Web information system.

Each concern is described by following three main steps: (1) Definition of a task taxonomy, (2) description of task performance and (3) specification of information requirements. Next, we introduce a brief description of each step. See [11] for more detailed information.

Step 1: Definition of a Task taxonomy. We define the different tasks that support the requirements of a concern in a task taxonomy. To do this, we consider a concern to be a task and then we perform a progressive refinement in order to obtain more specific tasks (see the upper left side of Figure 1). Furthermore, we can specify

temporal relationships between subtasks for ordering them according to a specific logic. To represent these temporal relationships, we use those ones introduced by the CTT approach [15]. The upper left side of Figure 1 shows the task taxonomy which supports the requirements of the concern *Collection of CDs*. According to this taxonomy, in order to collect products (root task *Collect CD*) users must search them (subtask *Search CD*) and next (the temporal relationship []>> implies a sequence of tasks) user must add them to the shopping cart (subtask *Add CD*).

Step 2: Description of Tasks Performances. We describe how users must perform each task defined as a leave node in the task taxonomy. To do this, we propose a technique based on the description of the interaction that users require from the system to perform each task. This type of description is done by using UML activity diagrams. In each activity diagram we specify (see the lower left side of Figure 1): (1) the actions that the system must perform in order to correctly support the task (nodes depicted by dashed lines). And (2) a set of Interaction Points (IPs) (nodes depicted by solid lines) that represents the moments during a task where the system and the user interact to each other. In each of these interactions, either the system provide the user with both information and access to operations (IPs stereotyped with the keyword «output») or the user introduce information into the system (IPs stereotyped with the keyword «input»). The information and operations that are provided in each IP are related to a specific entity. Furthermore, for each Output IP, the number of information instances that it includes (cardinality) is depicted as a small circle in the top right side of the primitive.

Figure 1 shows (in the lower left side) the description of the task Search CD. The task Search CD starts with an Output IP where the system provides the user with a list (cardinality *) of music categories. From this list, the user can select a category. If the category has subcategories, the system provides again the user with a list of (sub) categories. If the selected category does not have subcategories the system informs about the CDs of the selected category by means of another Output IP. From this IP, the user has two possibilities to continue the task: (1) to select a CD, and then the system provides the user with a description of the selected CD or (2) to activate a system action which searches for the CDs of an artist. To do this search, the user must introduce the artist (the search criterion) by means of an Input IP. If the search returns only one CD, the system provides the user with its detailed description. Otherwise, the system provides the user with a set of CDs.

Step 3: Specification of Information Requirements. We describe the information that the system must store in order to correctly support the requirements associated to each concern (e.g. the information that the system must store to allow users to correctly collect CDs). To do this, we associate an information template to each entity identified in the task performance descriptions. These information templates are inspired by techniques such as CRC Cards [4]. 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. We provide a name and a description for each feature. In addition, we use these templates to indicate the information exchanged in each IP. We indicate the IP/s (if there are any) where each feature is shown (Output) or requested (Input). To identify an IP, we use the following notation: *Output (Entity, Cardinality)* for Output IPs, and *Input (Entity, System Action)* for Inputs IPs.

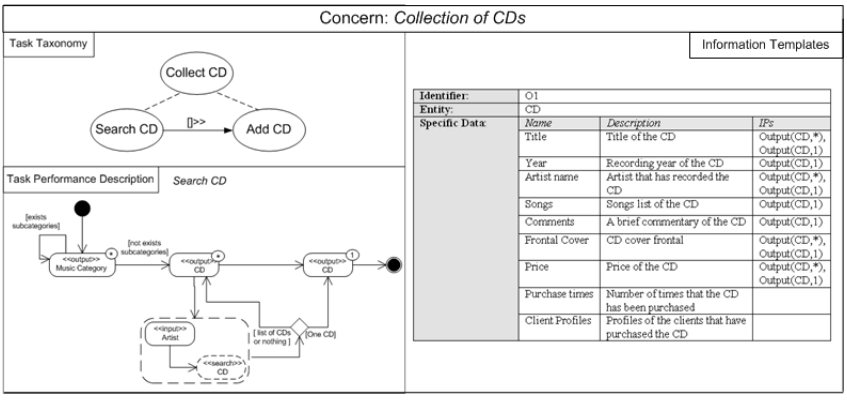

Fig. 1. Partial Description of the concern Collection of CDs

For instance, according to the template in Figure 1, the information that the system must store about a CD is (see the specific data section): the CD title, the artist's name, the front cover, and the price which are shown in the IPs Output(CD,1) and Output(CD,*); the recording date, some comments about the CD and the list of songs which are only shown in the IP Output(CD,1); and finally, the number of times that a CD has been bought and the profiles of the customer which usually purchase it are also stored. These two last features are not shown in any IP.

2.2 Obtaining Web Application Prototypes from Task Descriptions

In this section, we describe the OOWS strategy that allows us to automatically obtain Web application prototypes from task descriptions (see Figure 2).

According to Figure 2, a model-to-model transformation is applied first in order to derive a Web application conceptual model from the task-based requirements specification [11]. These transformations are defined by means of graph transformations. Graph transformations are rewriting rules that are made up of a Left Hand Side (LHS) and a Right Hand Side (RHS). They are applied as follow: When the LHS is found in a host graph, it is replaced by the RHS.

The obtained conceptual model is defined by means of the OOWS method [8]. The OOWS method proposes several models in order to describe the different aspects of a Web application: The system static structure and the system behaviour are described

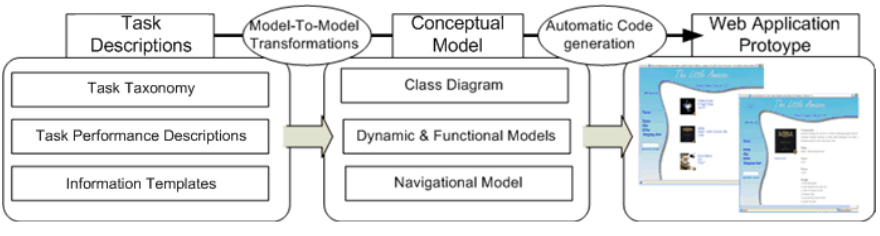

Fig. 2. The OOWS development process

in three models (*class diagram* and *dynamic*-and *functional* models) that are borrowed from an object-oriented software production method called OO-Method [2]. The navigational aspects of a Web application are described in a *navigational model*.

Next, a strategy of automatic code generation is applied to the conceptual model in order to obtain code. By following directives specified in design templates [8] a Web application prototype is automatically generated by the OOWS case tool [6].

3 Our Approach in a Nutshell

In this section, we explain how requirements of crosscutting concerns can be integrated. Three kinds of integration are proposed according to the elements used to perform it: (1) Integration at task taxonomy level, (2) Integration at task performance level and (3) Integration at task information level.

3.1 Integration at Task Taxonomy Level

We use the task taxonomy to perform the integration. To do this, we consider concerns to be tasks that can be connected to the task taxonomy of other concerns. For instance, let's consider the concern *Inspection of the Shopping Cart* which involves those requirements that allow users to inspect their shopping cart. We want to integrate the requirements involved in this concern with the requirements involved in the concern *Collect CDs*. In particular, we want that when users are collecting CDs they have always the possibility of inspecting the shopping cart. In order to do this integration, we connect the concern *Inspection of the Shopping Cart* (considering it to be a task) to the task Collect CD defined in the task taxonomy of the concern *Collect CDs*.

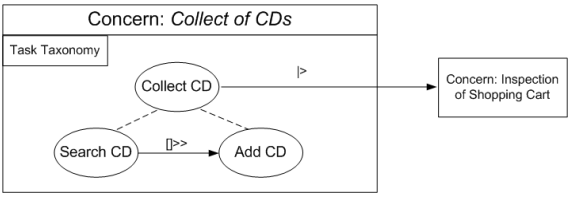

Fig. 3. Example of integration at task taxonomy level

Figure 3 shows this integration. As we can see in Figure 3, the temporal relationship that has been used to perform this integration is Suspend/Resume (|>). We use this relationship because its semantics (see step 1 explanation in Section 2.1) allows us to perform the integration in the way that we need: according to the relationship semantics, users can interrupt the task Collect CD at any time in order to perform the task-considered concern Inspection of the Shopping Cart. Thus, while users are collecting CDs they are always able to inspect the shopping cart. The implementation results of the integration presented in Figure 3 are shown in Figure 4. We can see how the Web pages that support the task Search CD (subtask of the task Collect CD) allow users to inspect the shopping cart.

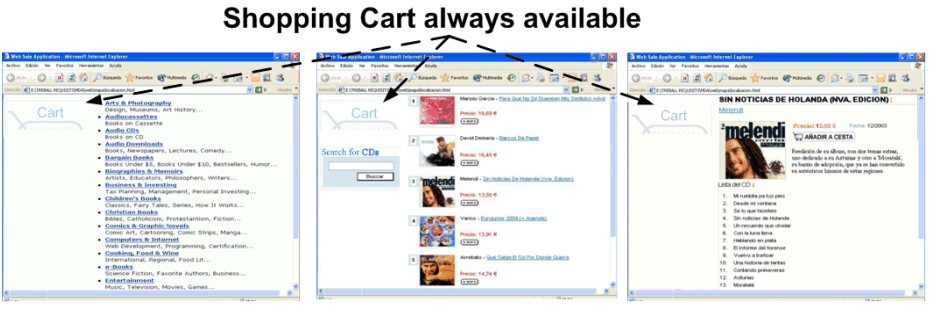

Fig. 4. Example of concern integration implementation

3.2 Integration at Task Performance Level

In this case, we use the description of a task performance in order to do the integration. This type of integration is based on the one presented in [9]. This type of integration allows us to connect an activity diagram node defined in the task performance description of a specific concern with a node defined in the task performance description of another concern. To do this, we propose the use of a set of composition rules which has the form:

```
Compose <Task_Base> with <Task_1, ..., Task_k>
{
 <Task_Base.Node>
 [Merge | AddConnection]  [to | with]
 < Task_i.Node>
}
```

By means of this type of rules we can indicate a base node defined in a task performance description of a specific concern (`<Task_Base.Node>`) and: (1) connect it to (`AddConnection to`) one or more nodes defined in the task performance description of other concern (`< Task_i.Node>`) or (2) merge it with (`Merge with`) one node defined in the task performance description of other concern.

For instance, Figure 5 shows the integration of the concept Collection of CDs with the concern *Playing of Video Clips*. This new concern involves those requirements that allow users to search the video clip of a song. Its description (task taxonomy and task performance description) is shown in the bottom left side of Figure 5. The task taxonomy of this concern is defined by an only one task. This task must be performed as follow: Users first access a list of video clips (IP Output(Video Clip,*)). From this list users can either select one an then access its description (IP Output (Video Clip,1)) or perform a search. The criterion of the search is introduced by the user throughout the Input IP. The results of this search are shown to users in a list.

The integration performed between these two concerns is the following: When users are collecting CDs and they access the list of CDs, users must have the possibility of search video clips. This integration is performed (see composition rule in the top left side of Figure 5) by connecting the IP Output(CD,*) defined in the description of

the task Search CD (concern Collection of Products) with the search system action defined in the description of the task Play Video Clip (concern Playing of Video Clips). The solid arrow in Figure 5 graphically shows the connection that the composite rule is defining. Figure 6 shows the implementation results of the integration presented in Figure 5. As we can see in Figure 6, the page that provides users with a list of CDs (which support the concern Collection of CDs) also provides users with the possibility of search vide clips. If users use this search, results will be shown in the Web pages that support the concern Playing of Video Clips.

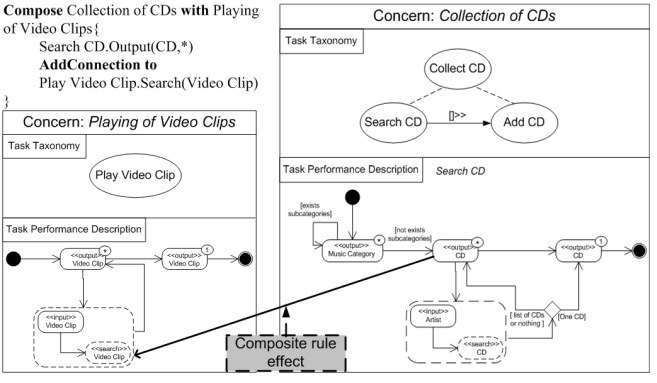

Fig. 5. Example of concern integration at task performance level

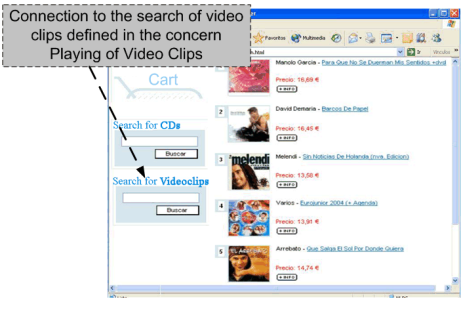

Fig. 6. Implementation result of the integration at task performance level

3.3 Integration at Task Information Level

We use this kind of integration when we want to extend some Output IP defined in a task performance description of a specific concern with information defined in information templates that belong to other concerns.

To do this, we use *information integration templates*. These templates are associated to an Output IP. They allow us to indicate that the IP must provide users with additional features that have been defined in an information template of other concern. In each information integration template (see Figure 7), we indicate an identifier, the IP that is

extended, and a feature section. In this section, we describe the new features that are incorporated to the IP. For each feature, we indicate the name, the feature's entity and the identifier of the information template associate to the entity, and the concern in which the information template is defined. Figure 7 shows an information integration template that allows us to integrate the concern Collection of CDs with the concern Playing of Video Clips. According to this template, the information that is shown in the IP Output(CD,1), which is defined in the performance description of the task Search CD in the concern Collection of CDs, is extended with the feature *media file,* which is defined in an information template of the concern Playing of Video Clips.

Identifier:	Id1		
IP:	Output (CD,1)		
Features:	*Name*	*Entity and Template*	*Concern*
	Media file	Video Clip, Id1	Playing of Video Clips
Population Filter:	Self.artist.name="Melendi" and Video Clip. song="Calle la Pantomima"		

Fig. 7. Example of information integration template

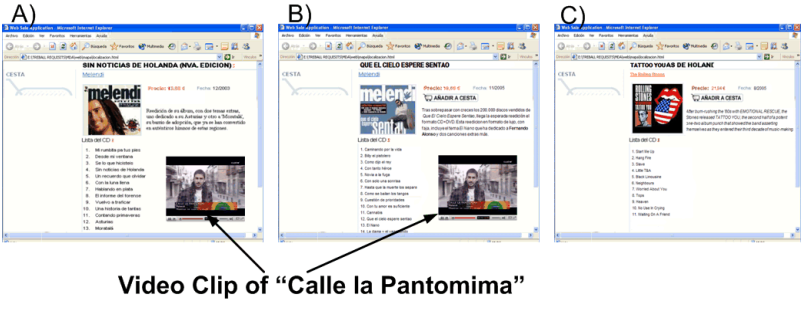

Fig. 8. Example of concern integration implementation at information task level

Furthermore, in each information integration template we can indicate a population filter (see the *Population Filter* field in the Figure 7 template). This filter allows us to restrict the information that is attached to the Output IP. It is defined by means of an OCL condition. The element *self* is used in this case to refer to the entity associated to the extended IP (CD in the example presented in Figure 7). The dot notation is used to access the different features defined for an entity. For instance, the information template in Figure 7 indicates that when users access the IP Output(CD,1), if the accessed CD is by the artist "Melendi", then the video clip of his song "Calle la Pantomima" must be shown. Figure 8 shows the implementation result of this integration. This figure shows three Web pages. Page A and Page B show information about two different CDs by the artist Melendi. In these pages we can see how the video clip "Calle la Pantomima" is shown. Page C shows information about a CD by The Rolling Stones. In this case the video clip is not shown.

4 Concern Integration in Prototyping Activities

In this section, we introduce how the integration of concerns defined in Section 3 can be taken into account in the process of generation of Web application prototypes (see Section 2.2). Two strategies can be followed to do this:

1. Perform the integration in the model-to-model transformation. We must extend the model-to-model transformation in order to interpret the concern integration defined in the task-based description and then derive the proper conceptual elements to support it. In this case, we obtain a unified conceptual model that supports the requirements of each concern already integrated. With this solution the generation of code from the conceptual model does not need to be changed.
2. Perform the integration in the automatic generation of code. In this case, we must: (1) Apply the model-to-model transformation for each concern. This provides us with a set of partial conceptual models that support the different concerns independently to each other. (2) Define the concern integration at the conceptual level. We must use a mechanism that allows us to define the same integration defined in the task descriptions but, in this case, in the conceptual model. Furthermore, we must define the proper transformation in order to automatically do this. (3) Finally, we must extend the strategy of automatic code generation in order to interpret the integration defined in the conceptual model and then generate the proper code. This solution allows us to perform the integration of concerns in later stages of the development process.

Due to the characteristics of the OOWS method, which does not provide mechanisms to define concern integration at the conceptual level, we have initially chosen the first strategy. In fact, we are currently extending the model-to-model transformation used in the prototype generation process in order to consider the integration of concerns presented in Section 3. However, we left as further work the development of the second strategy with a Web engineering method such as OOHDM, which provides us with mechanisms to define at conceptual level concern integration.

5 Concluding Remarks and Further Work

We have presented a novel approach for the representation and composition of Web application concerns at the requirements engineering stage. Functional concerns are modeled using tasks, and then integrated by applying task composition operators. The integrated model can then be mapped into an OOWS conceptual model and a prototype can be generated using existing transformation-based tools. We have shown with simple examples of archetypical Web applications how this process proceeds from requirements into design and prototype generation.

We are currently working on several research lines: first we are improving the concerns composition language to support more complex cross-cutting behaviors; besides we are exploring the generation of partial OOHDM models from concern models to experiment with composition at the conceptual and/or navigational levels.

References

1. Araújo, J., Whittle, J., Kim, D.: Modeling and Composing and Validating Scenario-Based Requirements with Aspects. In: Proceedings of the 12th International Requirements Engineering Conference, Kyoto, Japan (September 2004)
2. Pastor, O., Gómez, J., Insfran, E., Pelechano, V.: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems 26, 507–534 (2001)
3. Baniassad, E., Clements, P., Araújo, J., Moreira, A., Rashid, A., Tekinerdogan, B.: Discovering Early Aspects. IEEE Software 23(1), 61–70 (2006)
4. Wirfs-Brock, R., Wilkerson, B., Wiener, L.: Designing Object-Oriented Software. Prentice-Hall, Englewood Cliffs (1990)
5. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. Computer Networks and ISDN Systems 33(1-6), 137–157 (2000)
6. Valverde, P., Valderas, P., Fons, J., Pastor, O.: A MDA-based Environment for Web Applications Development: From Conceptual Models to Code. In: 6th International Workshop on Web-Oriented Software Technologies (2007)
7. Filman, R., Elrad, T., Clarke, S., Aksit, M. (eds.): Aspect-Oriented Software Development. Addison-Wesley, Reading (2004)
8. Fons, J., Pelechano, V., Albert, M., Pastor, O.: Development of Web Applications from Web Enhanced Conceptual Schemas. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, Springer, Heidelberg (2003)
9. Gordillo, S., Rossi, G., Moreira, A., Araujo, J., Urbieta, M., Vairetti, C.: Modeling and Composing Navigational Concerns in Web Applications. In: Proceedings of LA-Web 2006, IEEE Press, Los Alamitos (2006)
10. Güell, N., Schwabe, D., Vilain, P.: Modeling Interactions and Navigation in Web Applications. In: Laender, A.H.F., Liddle, S.W., Storey, V.C. (eds.) ER 2000. LNCS, vol. 1920, pp. 115–127. Springer, Heidelberg (2000)
11. Valderas, P., Pelechano, V., Pastor, O.: A Transformational Approach to Produce Web Application Prototypes from a Web Requirements Model. In: IJWET. International Journal on Web Engineering and Technology (2007)
12. Schwabe, D., Rossi, G.: An Object-Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems (TAPOS) 4, 207–225 (1998)
13. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, Springer, Heidelberg (1997)
14. Koch, N., Kraus, A., Hennicker, R.: The Authoring Process of UML-based Web Engineering Approach. In: IWWOST 2001. Proceedings of the 1st International Workshop on Web-Oriented Software Construction, pp. 105–119 (June 2001)
15. Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTree: A diagrammatic notation for specifying task models. In: Interact 1997, pp. 362–369. Chapman&Hall, Australia (1997)
16. Moreira, A., Rashid, A., Araújo, J.: Multi-Dimensional Separation of Concerns in Requirements Engineering. In: RE 2005. Proceedings of the 13th IEEE International Requirements Engineering Conference, IEEE Computer Society, Los Alamitos (2005)