Modeling and Composing Navigational Concerns in Web Applications. Requirements and Design Issues.

Silvia Gordillo[1,3], Gustavo Rossi[1,4,i] , Ana Moreira[2], Joao Araujo [2], Carla Vairetti[1], Matias Urbieta[1]

[1] Lifia. Facultad de Informatica. Universidad Nacional de La Plata, Argentina
[2] CITI/ dept. Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal.
[3]Also at CICPBA, [4]Also at Conicet
{gordillo,gustavo,carlav,murbieta}@lifia.info.unlp.edu.ar, {amm, ja}@di.fct.unl.pt

Abstract

Complex applications, in particular Web applications, deal with a myriad of different concerns and some of them affect several others. The result is that these crosscutting concerns are scattered throughout different software artifacts and tangled with other concerns. In this paper we present an approach for modeling and composing navigational concerns in Web applications. By showing how to build partial navigation scenarios with user interaction diagrams, analyzing how they crosscut and defining corresponding composition rules, we add modularity to the requirements specification stage, facilitating reasoning about the requirements and a consequent tradeoff analysis to support informed decisions of on architectural choices Moreover, by focusing on navigation concerns during the early stages of applications development, we aim to address the impact of crosscutting concerns in design models, improve the discovering of meaningful design artefacts and improve traceability of design decisions.

1. Introduction and Motivation

Web applications must cope with a myriad of functional and non-functional concerns. Some of them are typical of "conventional" software while others are idiosyncratic of the Web (such as navigation). Yet, others are application or domain-specific such as payment, advising, advertising, etc.

The only way to deal with these concerns is to be able to correctly identify and modularize them, understand the impacts and trade-offs among them and the relationships between design artifacts that realize them.

[i] This paper has been partially funded by Secyt under Procject PICT 13623

When designers are faced with the multiplicity of application concerns typical of Web software, they have a small repertoire of abstraction and composition mechanisms (e.g. those provided by the underlying object-oriented approaches, e.g. [6, 8, 9]) to deal with those concerns separately. A possible consequence is that a design artifact (a class for example) might describe several disparate concerns, or the same concern may appear scattered through different components thus making evolution hard. We argue that in most cases, crosscutting concerns are responsible for the emergence of complex navigational structures and therefore they should be identified and handled earlier in development.

Consider the simple example shown in Figure 1, part of the Amazon.com website, where information, relationships and operations corresponding to different concerns are present: the *"Product Information"* concern, marked with ellipse (1), allowing to obtain the basic product information (e.g. name, list price); the *"Marketplace"* concern at the right hand side for letting the user sell his products (2); the *"Gifts"* concern allowing to manipulate wish lists (3); the *"Advising"* concern showing related CDs (4); the *"Navigation History"* concern on the left hand side, giving information on the recently viewed items (5), and the *"User Record"* concern on top allowing the user to have information about his order history (6).

It is clear from the example above that the conceptual and navigation classes that abstract the CD page (e.g. using OOHDM [9] or UWE [6]) encompass information and functionality pertaining to different concerns. Notice also, that many outgoing links from this page correspond to different concerns, i.e. they might not be elicited just analyzing the basic product information needs.

Given the dynamic nature of web applications, techniques for advanced separation of concerns appear

to offer mechanisms that are worth being explored in this application domain.

Figure 1: Different concerns in the same page

This paper shows how the ideas of advanced separation of concerns can be applied to improve requirements specifications and early design stages of Web applications. It presents an approach that allows the early capture of crosscutting concerns that affect navigation and their representation using separated analysis artifacts. It helps to produce a modular design model, in which aspects might, or might not, be used.

The contributions of this paper are twofold: (i) offer a requirements approach to handle navigational concerns, characterizing the situations in which crosscutting exists; (ii) provide support for improving the construction of design models in complex applications. For conciseness, we base our discussion on the OOHDM [9] design approach, though the same ideas can be easily used by other object-oriented design methods, such as UWE [6] and OOWS [8].

The rest of the paper is structured as follows: Section 2 introduces navigational concerns and the core of our approach. Section 3 discusses how to improve design models. Section 4 addresses navigational models and Section 5 concludes and provides some future research directions.

2. Identifying and Composing Navigational Concerns

A concern refers to a feature that the future system needs to address to satisfy the stakeholders' needs. Also, according to [7], a concern implies any coherent set of requirements, e.g. all requirements referring to a particular theme or behavioral application feature.

We define a navigational concern as an application concern that impacts in the way users navigate the application; a navigational concern is reflected in some navigational information structure (e.g. a web page, a link), or behavior (e.g. checking if the user is allowed to navigate to a page). As a consequence, a navigational concern will also impact in the application's navigational model, either in a node or link class. These concerns may be realized in classes, methods, attributes (such as information or anchors), or links.

For example, in order to satisfy the requirements in the *Advising* concern of Figure 1, we enrich the CD page with links pointing to other related CDs. Certainly, as nodes and links are derived from conceptual classes and relationships, navigational concerns (which certainly are application concerns) impact in the conceptual model. In this example, we might need to provide corresponding behaviors to calculate the related CDs using the appropriate algorithms. Notice that these behaviors do not correspond to the core of the CD class, and they certainly might vary at a different pace with respect to other CD responsibilities.

Navigational concerns are critical in Web applications as the most important design decisions in these applications are related to the information they provide, their linking structure and the operations they support while the user navigates the information space.

We say that navigational concerns crosscut when they are scattered throughout navigational classes (i.e. the same concern shows itself in different node or link classes), or when they are tangled (i.e. the same navigational class supports more than one concern). Crosscutting concerns may appear as information or behavior that seems not to completely belong to the current node, as links that "break" the current task flow, as conditions that have to be checked in different classes or methods, etc. Concerns shown in Figure 1 are examples of crosscutting navigational concerns. They all crosscut the same software artifact (a CD page).

Notice that our definition of crosscutting is broader than the typical definition in aspect-oriented software design, where tangling has the flavor of code tangling. Therefore, the information we can derive from the fact that some concerns crosscut other concerns not only implies the definition of aspects, but also the identification of other design structures such as links, or other kinds of module compositions as we will show later in the paper. Early identification and description of navigational concerns help stakeholders to decide about potential trade-offs, improves application modularity and provide traceability to requirements, thus simplifying maintenance.

2.1. From meta to application specific concerns

The model we envisage to deal with navigational concerns at the requirements level is illustrated in Figure 2 and has been inspired by [7].

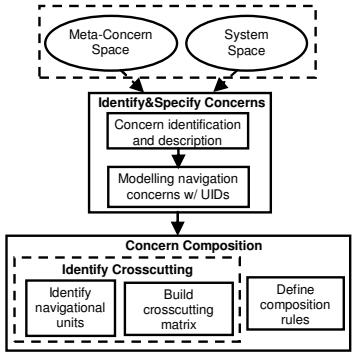

Figure 2: Overview of the approach

Certain concerns, be functional or non-functional, appear once and again during system development. This was the drive to propose a reuse mechanism with a catalogue of concerns defined in a very abstract and system-independent fashion. This catalogue, here named *meta-concern space*, should be used to help eliciting the problem domain concerns. The idea is to use these meta-concerns, to help organizing the system space requirements (where the application domain is described), so that concerns of the problem domain are identified. These concerns can be seen as specializations of concerns at the meta-space level. Meta-concern and system spaces are represented by ovals in Figure 2. Both the abstract concerns in the meta-concern space as well as their concrete realizations are specified using a set of well-defined templates based on the eXtensible Markup Language (XML). A catalogue of non-functional concerns has been provided in [2]. Here, we extend the notion of such a catalogue to typical navigational concerns found in web applications, such as Shopping, Advising, Sales, Navigation History, etc.

In summary, the system space contains the problem domain description we want to develop, obtained from different sources (interviews, ethnographic studies, analysis of business practices, etc.) and the meta-concern space contains a set of typical abstract concerns which repeatedly manifest themselves in various application domains. The identification of the concrete concerns can be achieved iteratively and incrementally, by handling small parts of the problem domain at a time. Note that not all concerns in the meta-concern space are necessarily used during this categorization; mostly only a subset of concerns relevant to the problem domain is needed.

Some of the most important concrete concerns (from the catalogue of abstract concerns in the meta-concern space) we have identified for our Amazon case study are:

- *Product Information*: to access basic information of the store products using different access strategies.
- *Shopping cart*: to handle the user's selection of products to be bought.
- *Authentication:* to check personal information against a valid *username* and *password*.
- *Navigation history:* to access to past information on previous user choices.
- *Advising:* to suggest alternative and potential products of interest.
- *Gifts:* to add, remove or update information about, for example, the users' personal wish list, wedding list, gift certificates, etc.
- *Checking out:* to handle the payment of the purchases.
- *Sales:* to inform users about sales and handle respective discounts during purchasing.
- *Donation*: to collect funds to be donated to particular charity organizations.
- *Marketplace*: to allow users to sell his own products in the store.

As an example we show in Figure 3 one meta-concern, Information Retrieval and in Figure 4 one of its concrete concerns, Product Information.

```
<MetaConcern name="InformationRetrieval">
    <Description>The operation of accessing information from a
        computer system </Description>
    <Examples>Database retrieval, Multimedia retrieval</Examples>
    <Relationships> Availability, Mobility, InformationUpdate
        </Relationships>
</MetaConcern>
```

Figure 3: Information Retrieval meta-concern

```
<Concern name="Product Information ">
    <Requirement id="1">
        The system will be accessed to provide basic information (e.g.
        for a CD there should be title, artist name, price, availability,
        cover, genre, record company, list of songs with name and
        excerpt), of the store products
    <Requirement id="2">
        Different search strategies such as finding the product given
        the title or part of the title can be used to access the product
        information.
    </Requirement>
    <Requirement id="3">
        Products will be organized in categories according to musical
        genre
    </Requirement>
    </Requirement>
</Concern>
```

Figure 4: Product Information concrete concern

2.2 Specifying Navigational Requirements with UIDs

Once the description of the problem domain has been organized into concrete specializations of concerns from the meta-concern space we can model the interactions related to each concern. To do that, those requirements in each navigational concern, which involve user interactions, are modeled with User Interaction Diagrams (UID) [5].

Similar to use cases [5], a UID describes the exchange of information between the user and the system, without considering user interface or design issues. UIDs enrich use cases with a simple graphical notation, allowing eliciting partial navigation paths. UIDs are simple state machines wherein each interaction step (i.e. each point in which the user is presented with some information and either indicates his choice or enters some value) is represented as an ellipse, and transitions between interaction points as arcs. We extend the UIDs to allow state numbering, which will be used later when we discuss composition of UIDs. In Figure 5 we show the UID for the use case, "sell one's CD given its title", from the Marketplace concern.

Figure 5 depicts an excerpt of the UID for one of the requirements of the concern Marketplace, the possibility that the user sells his own CDs. First, the user identifies the CD to sell by searching. Once found, the user chooses one and selects the operation *Sell my CD*, following the transition. Afterwards he must enter the actual CD condition (new, used, etc.), and some optional comments. Afterwards, s/he has to enter the intended price, location of the user and shipping method (third ellipse). The store informs other contending prices. The user confirms his identity and finally (not shown in the figure) he enters additional information such as credit card, contact telephone, etc.

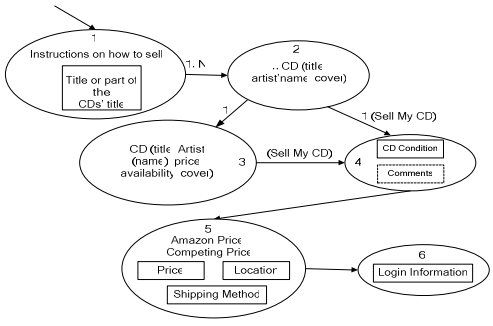

Figure 5: Selling a CD providing its title

The complete syntax for UIDs can be found in [5]. A UID is built for each use case involving meaningful interactions between the user and the system. UIDs provide a first insight about the information and linking structures that the application will manipulate, and an outline of the possible navigation flows that can be validated with stakeholders. Using simple heuristics described in [5]) conceptual and navigational models can be derived from UIDs.

Some navigational concerns, such as authentication, or adaptivity, will encompass requirements that do not have an associated user interaction, e.g. by implying a system check against the user identity (authentication) or some pre-processing before a link is traversed (adaptivity). In this case, there will be no associated UIDs, though these concerns are also analyzed regarding crosscutting.

2.3 Identifying and Characterizing Crosscuttings

Composition of concerns is then realized at the UID level and is achieved in two steps: (1) identifying navigational units and building a crosscutting matrix between navigational units and navigational concerns; (2) Composing UIDs of concerns which crosscut, analyzing how navigational units must be composed.

A navigational unit (NU) is the requirement counterpart of a navigational class (e.g. a node) in the navigational model and reflects an information structure that emerges from an interaction state in a UID (e.g. a CD, a shopping cart, etc). These information structures usually represent some data items presented to the user, and are generally the basis for performing the next interaction state in a UID, e.g. because some information is selected. A NU arises in a concern, which is called its dominant concern. Some requirements might give raise to more than one NU, such as the sequence of information structures arising during check-out and similar NUs might appear in different concerns.

In our example we derive a unit for Product Information (PI) and another for Marketplace (MP), both representing some information of the product to be bought or sold. Other units (whose UIDs are not shown) are: SC (shopping cart unit), NH (navigation history), CO (check-out unit), S (sales unit), etc. UIDs for some concerns such as Advising or Authentication might be too simple and thus we will not derive navigation units from them.

In our example, when accessing the information of a CD (the CD NU), we may wish to have an operation or link for selling our copy of the CD (the Marketplace concern) or to other recommended CDs (Advising concern), meaning that the NU is tangled with the

concerns represented by the outgoing links. Product Information is the dominant concern of the NU, and Marketplace (or Advising) is the crosscutting navigational concern. A closer look at UIDs in the Product Information and Marketplace concerns will reflect the fact that the central information units are similar or identical. Slight differences among the CD while accessed in different concerns (e.g. Marketplace and Product Information) might give a cue to designers for using advanced design structures (such as aspects, themes [3], or roles [10]), as discussed in Section 3.

Crosscutting is represented in the form *Navigational Unit* X *Concern*. For each navigational unit we analyze if it needs to provide information or operations pertaining to other navigational concerns, i.e. if the navigational concerns are accessible from or impact in that navigational unit. This is marked with a tick in the corresponding cell. A crosscutting concern is defined as a concern that is scattered among several navigation units. This means that the navigation units are tangled, since they include the properties of the dominant concern that originated them, as well as the properties of this new navigation concern.

For example, it is plausible that facilities related with Advising can be reached from units that implement Product Information, Shopping Cart, Gifts, etc.

2.4 Composing UIDs

The second step is to analyze the impact of crosscutting concerns on navigational units. We do so by composing those UIDs which reflect these concerns. Navigational units serve as the base for our compositions. In some cases the identification of crosscutting navigational concerns will not imply the composition of UIDs but meaningful information to be used in subsequent development activities (i.e. for those crosscutting involving concerns not represented by UIDs such as authentication or adaptivity). We have defined a set of composition rules which allow creating composite UIDs. To define composition structures, we extended the UID notation to include an identifier number in each state which is then accessed by using the "." notation as: *<UID_name . state_id>*

To access a transition, we use the specific operators *target* and *source* applied to states as: *Target(UID_name.state_id)*. To access a state element we can also use the dot notation: *UID_name.state_id.elementName*. On top of this, we need several operators: *AddConnection* to add a link to a UID state, *AddOperation* to add an operation to a state or a state element, *Merge* to merge two UID's states (with the semantics of set union) and *AddTransition* which creates a new transition, connecting two states from different UIDs. Any composition rule has the form:

Compose <UID_Base> **with** <UID_NavC$_1$, … UID_NavC$_k$>
{<UID_Base, UID_Base.State>
  [**Merge | AddTransion | AddConnection | AddOperation**] [**to | with**]
    <UID_NavC$_i$.State, UID_NavC.Operation, NavC$_i$>}
}

In Figure 6 we show the composition between the Product Information and Marketplace concerns, which is defined in the following composition rule:

```
Compose Product Information with MarketPlace
{ ProductInformation.3 AddConnection to
          Marketplace}
```

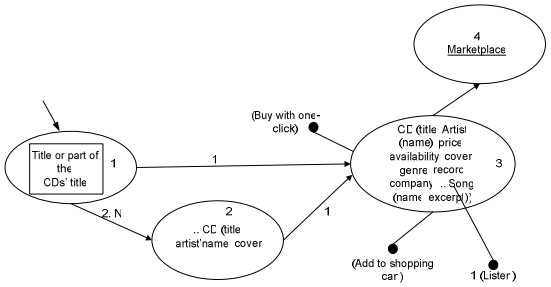

Figure 6: A connection to Marketplace added to Product Information

Notice that at this level of abstraction we are only showing that there is a connection between both UIDs. If, on the other hand, we want to see the full picture, by merging the navigational unit and navigation concern UIDs together, a different composition would be required:

```
Compose Product Information with MarketPlace
{ ProductInformation.3 Merge with
MarketPlace.3}
```

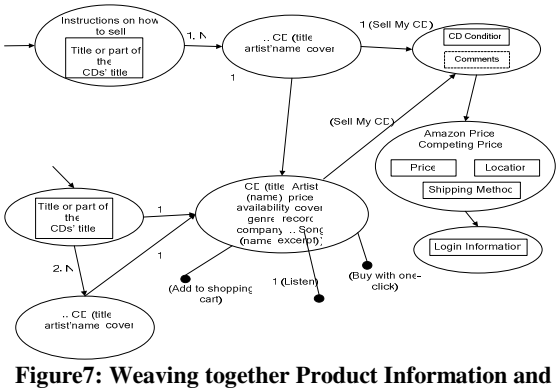

Figure7: Weaving together Product Information and Marketplace

The resulting UID is shown in Figure 7. Notice that the merge operation has yielded a navigation unit (for

CDs) which results from the composition of the two different views (Information and Marketplace). Besides, the composed UID has two entry points, corresponding to the components UIDs's entry points. A consequence of this finding can obviously be the need to provide a new interaction state in which the user decides if he wants to sell or buy.

3. Improving Design Models

As discussed in the previous section, the information gathered while analyzing crosscutting navigational concerns and their compositions is useful to enhance (or even define) the application's design models. Next we give an overview of the kind of improvements that we can obtain from the previously proposed requirement model.

- Discovering hidden relationships (which are derived from crosscutting information). For example, the fact that the Product Information and the Sales concerns crosscut means that we will have a relationship between the product and another related product in sale (e.g., the "Better Together" option in Amazon).
- Finding new operations. The example above (Product Information crosscutting Sales) might need an operation similar to *BuyBoth* in Amazon which does not emerge trivially by analyzing individual UIDs. This operation will be allocated into the conceptual class Product. Another example is the crosscutting between Product Information and Gifts which indicates that the class Product must support an operation to add the product to a Gift list as shown in Figure 1. Notice that this behavior is quite simple and can be easily generalized to different lists by just adding a parameter but can only be discovered by understanding the interweaving between these two concerns.

Sometimes, when a concern crosscuts another one it might mean that to perform an action (or to show information) corresponding to the former concern we need to check if some condition, belonging to the latter is satisfied. The most usual example of a crosscutting concern is Authentication. For example, to initiate the checkout, the user must be authenticated; analogously, to enter a review we need to check the user's identity. In this case, the best design solution is to encapsulate the authentication behavior in an aspect and then weave it [4] into the corresponding concern. Notice that if new concerns arise whose dominant NUs are crosscut by Authentication, we could simply weave the authentication behavior into corresponding modules by just tracing requirement information forward to design.

Roles [10] meanwhile can be identified by analyzing the variability of similar NUs when involved in interactions in different concerns. For example, products exhibit different features when accessed in the Product Information concern or in the Marketplace concern. While it makes no sense to derive completely different classes from the corresponding NUs (i.e., only one class supporting Products will be created), we can improve the design of class Product by defining two different roles: ToSell and ToBuy. Each role implements the corresponding variability and, at the same time, allows us to indicate which products can not be sold by users, i.e. they can not play the ToSell role. For example it is not reasonable that users sell food (e.g. the Gourmet Food store in Amazon is not involved in a Marketplace).

Finally, the same conceptual class may have to implement different themes which is indicated by a crosscut that implies the addition of many non-trivial operations; for example when requirements corresponding to the Advising concern (which crosscut Product Information) have to be implemented, we could just add operations which implement these requirements into the corresponding class (Product). However, the effect of this design choice is that we are polluting the class with methods that vary in a different pace (e.g. the recommendation criteria changes often or new kind of recommendations are implemented). A better solution is to create a separate design artifact, called a theme in [3], and later weave it into Product. This solution is more modular and requires less computational effort than aspects in the weaving process (as it might only require the juxtaposition of operations).

4 Improving Navigational Schemas

UIDs have been also used to help the designer derive the navigational schema from the conceptual schema. By analyzing different compositions of UIDs, as discussed in Section 2.4, we can obtain additional information on suitable navigation design structures. The most important ones are:

- Finding hidden links and therefore improving also the conceptual model with new relationships or operations. For example, the fact that Advising crosscut Shopping Cart indicates that we must implement links from the shopping cart to recommended products (e.g. the "See more items like those in the Shopping cart" option in Amazon).

- Preventing the definition of "dangerous" links. The fact that the Checkout NU is not crosscut by most concerns, indicates that while checking out we should not provide further links (e.g. to avoid reaching inconsistent states when abandoning the checkout process). This absence of crosscutting might appear as an explicit business rule or might have to be derived by closely analyzing Navigational Unit X Concern table
- Discovering "pure" navigational aspects, i.e. those which apply to the navigation and not the conceptual schema (as in [1]). An interesting example is the Navigation History; as the process to compute the history and the behavior to show it in the current page is completely located in the navigation classes (scattered through all nodes and also tangled), we can improve the modularity of the navigation model by defining a navigational aspect: Navigation History, and weave it into the final design model.

5 Concluding Remarks

In this paper we have presented a novel approach for modeling, analyzing and composing navigational concerns in Web Applications. While separation of concerns has been identified as a key strategy to improve modularity, it has not been used to model Web applications requirements so far. Similar to [7] we propose to group requirements that refer to the same application theme in concerns, by instantiating a meta-concern space. User Interaction Diagrams are later specified and concern crosscutting analyzed. Composition rules help to understand the interplay of these concerns and facilitate the interaction with stakeholders to early requirements validation. Our approach also provides better information for designers to build modular and evolvable design models. The impact of early concerns and concerns crosscutting identification in the web application development process can be summarized as follows:

- Improved separation of concerns (in this particular case, navigational concerns);
- Incremental (and iterative) development through incremental composition;
- Early identification of navigational concerns and crosscutting and tradeoff analysis before an architecture design is derived;
- Better identification of design model features, such as non obvious links, roles and aspects.

We are currently conducting research on:

- Extension of the approach to handle other typical crosscutting concerns, such as security, availability, synchronization, etc. We are analyzing the interplay between navigational and application concerns, for example in order to derive non navigational crosscutting; as a consequence we are researching in how the information gathered during the process of UID composition can be used to improve application modeling.
- Improvement of the UIDs semantics to allow the identification of every single element therein to be able to define a fully fledge composition language for UIDs.
- Formal definition of the heuristics which allow improving design models, extending the ones defined in [5].
- Completion of our software tool to cover the whole life cycle and acting as a concerns' baseline, providing traceability forward and backwards of design decisions.

6. References

[1] Baumeister H., Knapp A., Koch, N., and Zhang G. Modelling Adaptivity with Aspects. In Proceedings of the 5th International Conference on Web Engineering (ICWE2005) (Sydney, Australia), Springer Verlag LNCS.

[2] Chung, L., Nixon, B., Yu, E., Mylopoulos, J. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, 2000.

[3] Clarke, S., Baniassad, E. Aspect-Oriented Analysis and Design. The Theme Approach. Addison-Wesley, Object Technology Series, 2005. ISBN: 0-321-24674-8

[4] Filman, R., Elrad, T., Clarke, S., Aksit, M. (eds.). Aspect-Oriented Software Development. Addison-Wesley, 2004.

[5] Güell, N., Schwabe, D., Vilain, P. Modeling Interactions and Navigation in Web Applications. ER (Workshops) 2000. (Utah,USA, October 2000) 115-127

[6] Koch, N., Kraus, A., Hennicker R. The Authoring Process of UML-based Web Engineering Approach. In Proceedings of the 1st International Workshop on Web-Oriented Software Construction (IWWOST 01), (Valencia, Spain, June 2001) 105-119

[7] Moreira, A., Rashid, A., Araújo, J. Multi-Dimensional Separation of Concerns in Requirements Engineering in Proceedings of the 13th IEEE International Requirements Engineering Conference (RE 2005). (Paris, France, August 2005). IEEE Computer Society.

[8] Pastor, O., Abrahão, S.M., Fons, J. An Object-Oriented Approach to Automate Web Applications Development in Proceedings of EC-Web 2001. (Munich, Germany, September 2001) 16-28.

[9] Schwabe, D., Rossi, G. An Object-Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems (TAPOS). Vol 4 (1998) 207-225

[10] Steimann, F.: On the Representation of Roles in Object-Oriented and Conceptual modeling. Data and Knowledge Engineering 35 (2000) 83-106