Using meta-level constructs in Web personalization

Sofie Goderis _, Gustavo Rossi _, Andres Fortier _, Juan Cappi _, Daniel Schwabe _

_ Programming Technology Lab, Vrije Universiteit Brussel

_ LIFIA-Facultad de Informatica-UNLP, Argentina

_ Depto de Informatica, PUC-Rio, Brazil

Abstract

In this position paper we analyze the problem of Web applications personalization from a design point of view. We focus on which design constructs are necessary to achieve modular and evolvable personalized Web Applications. We claim that personalization involves different concerns (e.g. rules, profiles, etc.) that should be clearly identified and decoupled; we show how to add personalized behaviors to existing applications in a non-intrusive way, by using reflective mechanisms found in most object-oriented languages.

We first introduce our approach to Web applications modeling that separates conceptual from navigation and user interface design; we next introduce personalization patterns and briefly show how they can guide the designer towards his objective. We finally present our approach and some ongoing research directions related with the design of an object-oriented framework for Web Applications personalization.

1-Introduction

Designing personalized Web applications implies dealing with different concerns such as building different interfaces according to user preferences or Web appliances, providing customized links; showing personalized information, adapting application's functionality, etc. To solve these problems we have to model the user, implement personalization policies and rules and integrate them into the application. Except from trivial read-only Web applications (such as www.my.yahoo.com), the process of personalizing existing Web software is a good example of complex applications' evolution. This is due mainly to the very nature of Internet software in which the rate of change (of requirements, marketing decisions, etc) is quite fast.

Though user modeling and profile derivation has been already discussed in the literature [Perkowitz 00], design aspects related with this problem have been seldom taken into account. The aim of this short paper is to present our conceptual framework for dealing with those design issues in a modular way, emphasizing concerns separation from requirement elicitation to application's implementation.

By clearly understanding and decoupling the design concerns involved in a personalized e-commerce application, we can keep the software manageable; we can also provide the basis for reusing designs and design experience. We have a better way to understand the kind of interactions appearing in a personalized application from an abstract point of view (independent of the specific aspects of the application), and to simplify them by following well-knwon design patterns. The structure of this paper is as follows: we first present our view of Web Applications design; then we introduce personalization patterns and show that all these patterns are finally mapped to objects' behaviors. We then explain our strategy for personalizing those behaviors using meta-level constructs. A discussion on possible implementations using a logic programming framework is finally discussed.

2-Our view of Web Applications design and Personalization

We follow the OOHDM approach to Web Applications design [Schwabe 98] The key concept in OOHDM is that Web application models involve a Conceptual, a Navigational Model and an Interface Model [Schwabe98]. Those models are built using object-oriented primitives with a syntax close to UML [UML00]. The concern of the conceptual model is to represent domain objects, relationships and the intended applications' functionality. In the OOHDM approach, users do not navigate through conceptual objects, but through navigation objects (nodes) that are defined as views on conceptual objects. As we consider Web applications as hypermedia applications, we define links connecting nodes, as views on conceptual relationships. Finally, the abstract interface model specifies the look and feel of navigation objects together with the interaction metaphor.

Using this approach as a conceptual framework to reason on personalization we have mined personalization patterns in existing Web Applications [Rossi 01a]. Next we summarize those patterns using a simplified format describing the problem and the corresponding solution. For the sake of simplicity we ignore interface personalization.

2.1 Link personalization

Problem:

Web applications involve dealing with a large number of objects and the way in which we reach them may depend on many different factors. We want to provide different users (individuals or roles) with different linking topologies.

Solution:

Personalize links by calculating the end-point of the link with user-related information. When we personalize links, all users access the same information objects and although anchors may look similar (see for example the link to Recommendations in www.amazon.com), each individual has a different customized topology.

2.2 Structure personalization

Problem:

Many applications involve not only dealing with thousands of objects but also with a great variety of subjects and services. We may want to circumscribe the navigation space to the aspects the user is interested in.

Solution:

Personalize (or let the user do it) the structure of the Web application. Consider the information space as a set of aggregated objects (or modules) and select only those objects that the user may want to consume as for example in www.my.yahoo.com.

2.3 Content Personalization

Problem:

In some Web applications we may want to provide each individual user with a slightly different content for a particular information item.

Solution:

Define personalized contents in nodes by letting node attributes vary according to the user; for example e-stores providing personalized prices for their products.

2.4 Behavior personalization

Problem:

Web applications combine hypertext navigation with other functionality (such as bidding, adding products to a shopping cart, etc). Suppose that we want to provide individualized responses to a particular operation

Solution:

Personalize the application behavior by making this behavior dependent of the user who triggers it.

In Figure 1 we summarize the previous discussion showing how the OOHDM approach deals with personalization in an abstract way. Notice that the previously presented patterns are independent of the method you use to build Web Applications: they exist on their own.

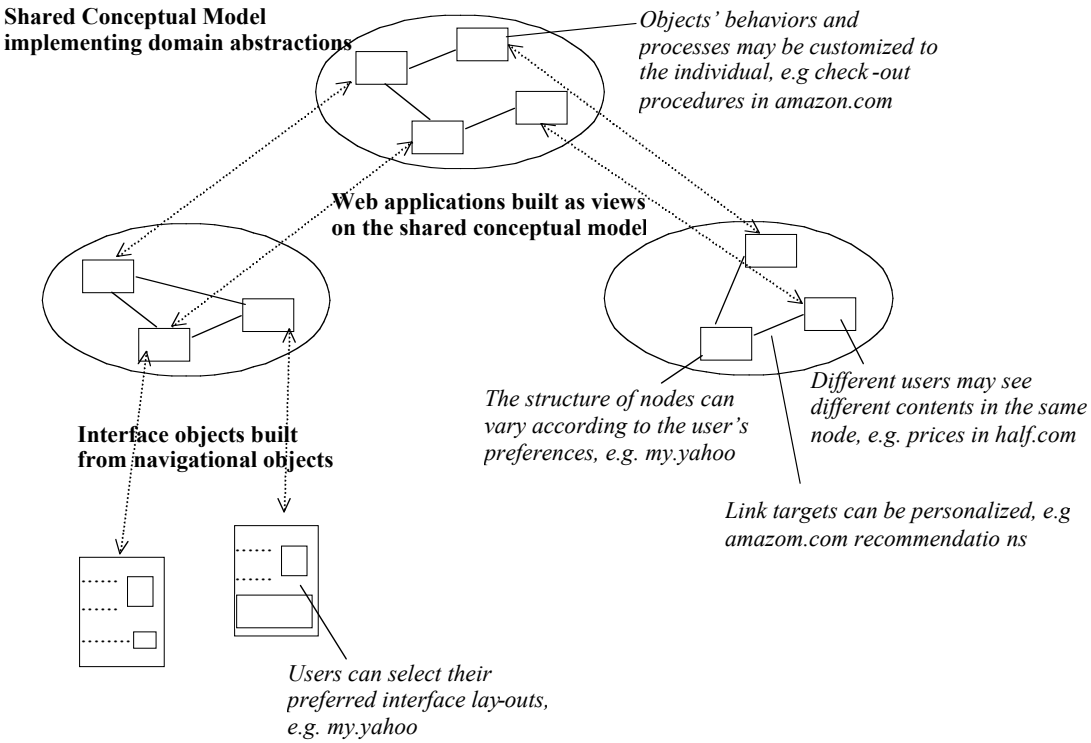

Fig. 1. Personalization in an OOHDM model.

Understanding previous patterns we can map the personalization requirements of our application to one of those patterns and then analyze how to deal with the design structures underlying those patterns. The aim of this paper is to analyze the process of building personalized applications focusing on those design aspects. In particular we want to show that correct design decisions impact positively on the applications' evolution, simplifying the process of obtaining personalized applications. A first approach for dealing with personalization in Web applications can be found in [Rossi 01b]

3-Design Issues and Concerns related with Personalization

It is easy to see that most (if not all) examples of personalization involve some kind of adaptable conceptual model. Therefore, understanding how to personalize a conceptual (application) model is essential for achieving personalized nodes,

links and behaviors. The field of adaptive software is not new (See [Oreizy 99]) though we are mainly interested in those design constructs that can simplify the process of adding personalization features to a Web application.

There are basically three concerns related with personalization: the user profile, the personalization rules and the application of those rules for a particular individual. Hard-coding personalization rules together with core application objects yields difficult to maintain software, because the pace of evolution of customization rules (and the associated user profiles) is faster than changes in the basic application model. Suppose for example that in an electronic store we have a Class Product with a method *price* to calculate its price. If we want to personalize the price such that we provide discounts to some customers (for example according to their buying history), we can modify the code of *price* such that it interacts with the user profile to get the information. However, each time we change the personalization rule we must change that method; the solution simply does not scale-up, since we keep tweaking the model's behavior to accommodate to the discount policies.

The same is true if we want to have different recommendation algorithms for different customers, or customized check-out procedure according to user preferences.

While variations in the domain model, such as adding new products or paying mechanisms can be solved by sub-classing and composing objects, changes related with personalization rules are more difficult to handle.

As will be described in deeper detail in the next section, the solution to this problem is not to decide which object in the domain model should be responsible for taking care of things like holding the algorithms or the policies, but to understand that everything that involves personalization must be dealt in another way. We think that by building a robust design in the meta-level we can achieve the exact degree of decoupling between the application's domain model and the personalization-related constructs, obtaining a scalable architecture.

4-Our Approach

The key of our approach is the recognition that those concerns related with personalization, namely the user profile, the rules and the application of rules, must be decoupled from the Web application model and further decoupled from each other. We next summarize our approach explaining the micro-architectural constructs we used for solving this problem:

4.1- Modifying the base application's behavior

There are basically two approaches for solving this problem: using decorators [Gamma 95] or intercepting behaviors with a meta-level approach. The latter solution clearly recognizes the fact that this problem should be treated as a separated aspect of the application and dealt with in an orthogonal way. We used an idea similar to method wrappers that "trap" the desired behavior and let us call the personalization code. The idea is that each time an object receives a message that should be personalized, we automatically "invoke" some functionality in the meta-level that "reasons" about the base application level and modifies the application's behavior. For example we can solve the price problem by intercepting the message *price* and provide a way to trigger rules that provide the personalized behavior. Moreover, we can do this in an instance basis (i.e. only some products have a personalized price).

When considering the price problem, we know in advance that each time the message price is sent, the price might have to be personalized. We can thus statically determine what personalization rule is to be used. However, in some cases determining what personalization rule is to be used, cannot be decided until runtime. This dynamic determination occurs for instance when personalization is different for different types of users. Making the distinction between static and dynamic determination of personalization will improve our approach significantly.

4.2-Implementing Rules

Rules should be treated as first class citizens in personalized applications. As they tend to evolve quickly and may be usually combined to implement complex business strategies, we must have a flexible approach for rules design. We are now experiencing with the SOUL meta-programming framework [Wuyts 01]. SOUL integrates Smalltalk and Prolog by letting designers write Prolog rules (in the meta-level) governing the behavior of the base level. Though originally conceived for synchronizing design and implementation, we have used SOUL to express personalization rules. Inheriting well-known Prolog features, rules are modular and easy to express. They can invoke base application's code and can be maintained efficiently, i.e. it is easy to add new rules or to modify existing ones as they are written in a natural way (*action* if *conditions*).

```
Rule updatePrice(?ResultString) if
isReceiver([Product],[#price],[?aProductObject]),
        currentCustomer(?ID),
        changePrice(?ID, [?aProductObject], ?ResultString).
```

Fig. 2. Personalization rule used for the price problem.

The rules specifies that if a certain Product object receives the message *price*, the personalization of this price, specified for the current customer, has to take place. The rule results in a piece of Smalltalk code, specifying how the price should be adapted, that will be executed by the Smalltalk base level.

4.3 Dealing with the user profile

The user profile is an important component in every personalized application.Facts about the user can be specified using Prolog or they may just be "traditional" objects. In the first case, the user profile is stored at meta-level and consists of a set of facts, such as those that we see next.

Fact name(id408, John).

Fact age(id408, 28).

Fact boughtProducts(id408, <BookID20, BookID403, CDID230>).

Accessing this data is simply done by using the right predicates. For instance, `name(id408, ?UserName)` will provide us with the name of the particular user with identity id408.

Another possibility is considering the user as an object on base level, but this will only influence the predicates used to access the profile (and not the personalization rules that use these predicates). For instance, accessing the user's name now requires the rule that we see below.

Rule name(?User, ?Name) **if**

equals(?Name, [?User getName]).

Such that the name is retrieved from the user object at base-level (i.e. `?User`). In our approach this is rather straightforward, because SOUL provides the necessary mechanisms to do so.

The rule-based system behind Prolog allows us to write inference rules to classify users according to the observed facts. In this way, we can associate algorithms to different types of users and write the code that assigns the correct algorithm to the current user as a set of logic predicates. These predicates may be executed each time we want to personalize an object's feature or with other strategies (once a day, each time we require it, etc). Keeping the user profile as a

separate module allows us to specify different strategies to support its evolution (both in structure and contents). For example, while for some applications, new facts about the user are added as a consequence of an application's operation (buying a product for example), it may be the case that we want to record the products he visited; with our meta-level approach we can just intercept the corresponding behavior (displaying a product for customer c) and add a fact to our information base (prolog predicates) in a transparent way.

5-Discussion and concluding remarks

We have briefly outlined our approach for personalizing Web application models. It is based in separating the main concerns behind personalization (rules, profiles, core model) and providing a design model for supporting this decoupling. Using a meta-level approach guarantees that we can seamlessly add personalized behaviors in a non-intrusive way, without affecting core application's functionality.

We are now studying how to provide generic meta-level classes and objects and templates for rules that can be used in different applications. The idea is to have abstract skeletons for personalization acting as application frameworks.

In this way, we will be able to personalize Web applications by just composing the base functionality with objects taken from some of these classes thus simplifying the task of adding rules, generating user profiles and applying the rules to the chosen application's objects.

References

[Gamma 95] E. Gamma, R. Helm. R. Johnson, J. Vlissides: "Design Patterns. Elements of reusable object-oriented software", Addison Wesley 1995.

[Oreizy 99] P. Oreizy, M. Gorlik, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, d. Rosenblum, and A. Wolf: "An architecture-based approach to self-adaptive software". IEEE Intelligent Systems, pages 54-62, May 1999.

[Perkowitz 00] M. Perkowitz, O. Etzioni: "Adaptive Web Sites" In Comm ACM, August 2000, p.p. 152-158.

[Rossi 01a] G. Rossi, D. Schwabe, J. Danculovic, L. Miaton: "Patterns for Personalized Web Applications", Proceedings of EuroPLoP 01, Germany, July 2001.

[Rossi 01b] G. Rossi, D. Schwabe, R. Guimaraes: "Designing Personalized

Web Applications", Proceedings of the 10th International Conference on the WWW (WWW10), Hong Kong, 2001,Elsevier, p.p.

[Schwabe98] D. Schwabe, G. Rossi: "An object-oriented approach to web-based application design". Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, v. 4#4, pp.207-225, October, 1998.

[Wuyts 01] Roel Wuyts: "A Logic Meta-Programming Approach to Support the Co-Evolution of Object-Oriented Design and Implementation" Phd Thesis. Programming Technology Lab, VUB, Brussels, 2001.