Modeling User Behaviour Aware WebSites with PRML

Irene Garrigós and Jaime Gómez

Universidad de Alicante, IWAD, Campus de San Vicente del Raspeig, Apartado 99
03080 Alicante, Spain
{igarrigos, jgomez}@dlsi.ua.es

Abstract. Adaptive websites usually change as effect of user navigational actions. Most current web engineering approaches (which consider personalization) allow to detect basic user browsing behaviour (e.g. user click on a link) but don't consider the definition of (complex) behaviour events or user behaviour pattern recognition. In this paper, we use a method independent language called PRML to specify behaviour aware websites. PRML evolved out the experience of OO-H and was designed to be a generic personalization specification method that can be reused for different web design approaches. PRML allows the personalization when a complex behaviour event is triggered (i.e. a sequence of links) and also allows the definition and recognition (at runtime) of user behaviour patterns.

1. Introduction

In the last decade, the number and complexity of websites and the amount of information they offer is rapidly growing. We face a new, wide spectrum of web applications that leads to new challenging requirements like the need for continuous evolution. Moreover websites typically serve large and heterogeneous audience what can lead to maintenance and usability problems [11].

Introduction of web design methods and methodologies [5] [6] [8] [9] [10] [12] have provided some solutions for designers (design support, help in determining consistent structuring, easier maintenance) and for visitors (better tailored content, easier navigation). In order to better tailor the site to one particular user, or a group of users, some methods provide personalization support. Usually web applications are adapted as effect of user browsing behaviour. Most current web design methods limit the detection of browsing behaviour to simple user navigational actions (i.e. start of a session, activation of a link…). Sometimes this leads to too simple personalization strategies definition.

To tackle the problems described above we provide a method-independent personalization specification language called PRML (Personalization Rule Modelling Language) [7] to build a (reusable) Personalization Model. This language allows explicit modelling of personalization policies and their effective deployment and reuse for different web design methods. PRML considers four

adaptivity dimensions: *the user characteristics* (e.g. user's age, user's hobbies, user's vision level…), *the user context* which includes different types of context like *device context* (e.g. PDA, PC, WAP), *time context* (i.e. date and local time of the connection), *network context* (e.g. latency, speed, bandwidth and *location context* (i.e. ubiquity of the user), the third dimension are *the user requirements* (specifying the needs and goals of the user) and the fourth is *the user browsing behaviour*.

If we focus on the *browsing behaviour* dimension we can see that the definition of complex navigational actions (i.e. sequence of link activation) is supported. Moreover it supports the definition and detection of user behaviour patterns. The consequence is that the complexity of the personalization strategies to define (based on user browsing behaviour) increases. The detection of user navigational patterns (defined for a very common user behaviour) at runtime implies some problems like how to track the user behaviour and how to recognize certain behaviour pattern. This is explained in section 4 of the paper.

Most web design methods [2] [5] [10] [14] and adaptive reference models have support for adaptivity considering (only) basic user browsing behaviour actions. The adaptive web engineering approach UWE [10] has some basic support for personalization based on user behaviour. The AHAM [14] reference model, which is mainly used for teaching applications, describes behaviour being tracked and used to update the user model. WSDM [5] supports adaptivity (i.e. not for a specific user) tracking the user browsing behaviour. WebML [6] is an exception supporting the definition of a personalization strategy based on complex behaviour user actions. It uses the WBM formalism based on a timed-state transition automaton to define the behaviour user actions that should be performed to trigger a personalization action. However, none of these approaches consider the detection (at runtime) of behaviour patterns.

The paper is structured as follows. In the next section, a case study defined in the context of the OO-H method is presented. The paper continues describing in section 3 the PRML fundamentals. How PRML is extended to support (complex) browsing behaviour and to detec behaviour patterns is presented in section 4. We continue in section 5, explaining the OO-H execution architecture supporting PRML rules. Finally, section 6 sketches some conclusions and further work.

2. Case Study: Blog of a University Lab

We are going to explain our approach in the context of the OO-H (Object Oriented Hypermedia) method [8] in which the PRML language was born. This is a user-driven approach based on the object oriented paradigm and partially based on standards (UML, OCL, XML…). OO-H allows the development of web-based interfaces and their connection with pre-existing application modules. In OO-H four views are defined: *the domain view¸ the navigational view, the presentation view* and *the personalization view.* Each of these views has associated a model (or

set of models). We are going to show the OO-H views by means of a case study excepting the presentation view that is not considered in this work.

The system to model is an online blog of a university lab in which (registered) users can basically post and consult messages (classified by categories) and set comments on them. In the left part of Figure 1 we can see the *domain model* (from the OO-H domain view) of the system. The right part of the Figure 1 shows the *user model* which will be explained further.

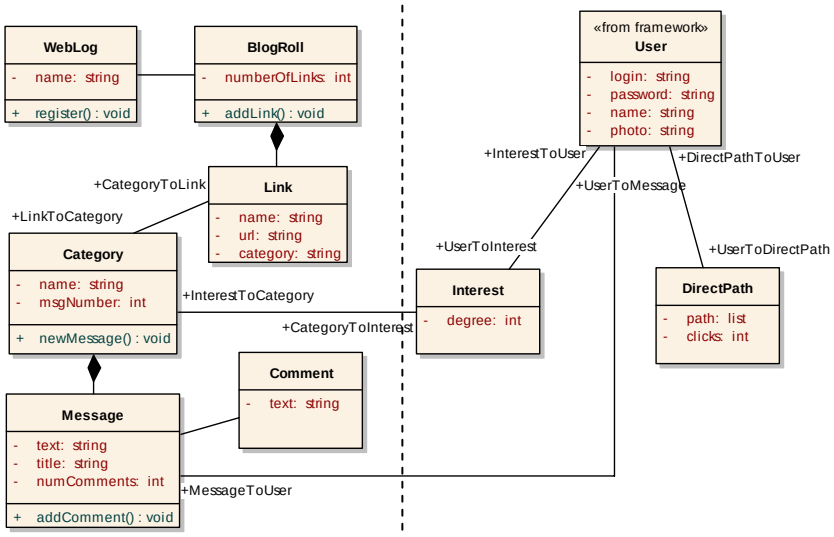

Figure 1. OO-H Domain and User Model

In the OO-H navigational view the navigation structure is defined by means of the *Navigation Abstract Diagram (NAD*). This diagram enriches the domain view with navigation and interaction features. In figure 2 we can see the (zero level of the) *navigational model* of the online blog system. The starting point (*entry point* element) is the home page where the user has to register. Registered users access to a page which has a collection of links (represented by an inverted triangle) in which there is a set of links to other related blogs (blogroll), there is also a set of categories and a set of messages. The links we can see in Figure 2 (ViewBlogRoll, ViewCategories and CheckMessages) are automatic links represented by a discontinuous line (i.e. the user does not need to click on them) and they show the information in origin (we can see that from the white arrow head). In Figure 2 we cannot see the whole NAD (for simplification reasons), we have two navigational targets which group modeling elements that collaborate to solve some functional requirements. A navigational target (NT) is represented by a UML package symbol. For a more extensive description of the NAD diagram see [8].

In Figure 3 we can see the details of the categories navigational target. In the menu page the user will find a set of categories. This set is an indexed list, so the user can click in one of the categories names to view the messages attached to it (the *Message* class). Moreover the messages can have attached comments. There are

also two method invocation links that allow to *add a new comment* and *add a new message* to a certain category.

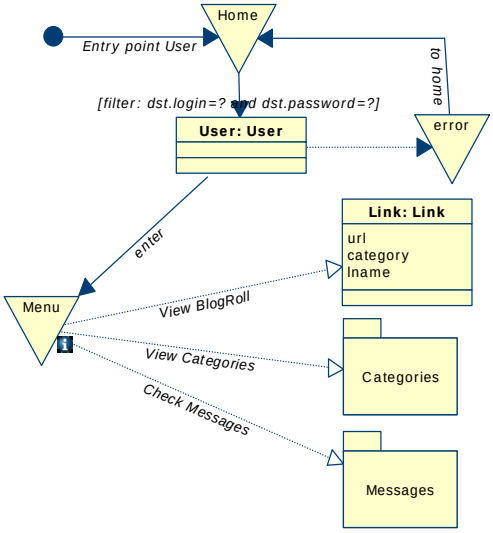

Figure 2. (Level 0 of the) NAD for the blog case study

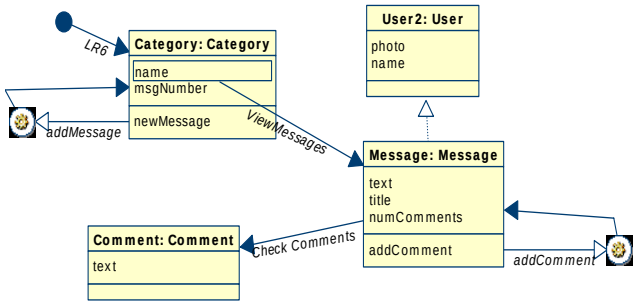

Figure 3. Categories NT

Due to space limitations, the details of the messages navigational target cannot be presented. It shows the whole set of messages with their corresponding data.
Finally, the *personalization view* is supported by a personalization framework that is part of the OO-H approach. This framework can be instantiated by the web designer and connected to any OO-H based website to which provide personalization support. It is divided in two parts: the *User model* and the *Personalization model.* They are explained next.

2.1 Personalization Support

The *user model* (UM) specifies the data (and its structure) needed for personalization (including user data). While a UM is conceptually a separate model, is used in a similar way as a *domain model* (DM): it is typically associated

with DM and can be represented as part of the DM. The UM of the system can be seen in the right part of Figure 1. To build this model we have to take into account the personalization requirements of the system to determine the (updatable) information needed to personalize. In the case study we consider the following (personalization) requirement:

- *Users will see the categories (for messages) sorted by (the user) interest on them.*

To cover this requirement in the updatable data space defined by the UM of the example we store the user interest degree on a category. For this purpose, in the UM we have the *Interest* class. To determine the user interest we use several strategies as we will see in section 4.
To store and update the information in the UM we need some mechanism. Also to specify the personalization strategies based on that information. For this purpose we need to define the *personalization model.* As we already explained, it is specified by a (method-independent) specification language which was born in the context of the OO-H approach. This language is called PRML (Personalization Rules Modeling Language) and its fundamentals are explained in next section.

3. PRML Fundamentals

PRML [7] is a modelling language for personalization that evolved out of the experience from the OO-H approach [6] [8] and was designed to be a generic personalization specification method. In PRML two conformance levels[1] are defined depending on the richness of the personalization specifications supported:

- In the first one of these levels, called basic (or zero) level, the (personalization) actions that can be specified have been defined by abstracting the basic operations supported by known web design methods [5] [6] [9] [10] [12]. The purpose of its definition is being able to specify a (universal) reusable personalization policy. This level is a subset of the advanced level.
- The second of the PRML conformance levels, called advanced level, comprise all the operations supported by PRML allowing the specification of richer personalization policies than the basic level (limiting though the reusability).

Moreover, the definition of these two levels allows to (easily) compare the level of personalization supported by the different methodologies because we have a unique way of specifying it (for all the methods). A methodology can have full or partial conformance with one or two of the levels, allowing then a better comparison.

Rules that can be defined using this language are ECA [4] rules. This decision is natural and conforms to initial requirements on a personalization model, where actions are taken as reactions on concrete events caused by users. The effectiveness of the concept of ECA rules for user-adaptation is proven by different personalization frameworks and tools (e.g. AHA! [14]). The PRML has its own language defining conditions and actions. An important part of the language is a

[1] This idea has been inspired by the SQL conformance levels.

mechanism specifying access to data attributes by means of path expressions that can be exported to different query languages for different data modeling techniques (relational, object-oriented, or semantic web modeling languages). This decision targets the requirement of independence on concrete web and data modeling methods. This is an important design requirement because it is obvious that the language should not contain any constructs dependent on concrete web or data modeling methods for being generic.

For satisfying a personalization requirement we often have to define how to update the knowledge about user (acquisition rule). This information is stored in the UM data space. Furthermore, we have to define the effects this personalization causes to the presented content and navigation structure by means of personalization rules (we do not consider presentation features in this work). These rules use the information specified by the UM to describe adaptation actions. A PRML rule is formed by an event and the body of the rule containing a condition (optional) and an action to be performed. The basic structure of a rule defined with this language is the following:

When *event* **do**
    If *condition* **then** *action* **endIf**
endWhen

When an event is triggered if the condition (optional) is satisfied an action is performed. PRML considers the following event types to trigger a rule:

- *Navigation event:* It is caused by the activation of a navigational link. Links have information about an instance or a set of instances they are associated with and are going to be shown in the activated node (i.e. the page resulting from the navigation). When the link is activated the rule/s attached to this event is/are triggered passing to the rule/s this information as a parameter[2] for personalizing the (adaptive) active node based on this parameter. We need the parameter value before the node resulting from navigation is generated (in order to use this value in the rules and build the adaptive web page). The parameter passed to the rule can be a simple parameter (when the data on the webpage is the instance of a concept of the DM) expressed in PRML as follows: "**When** *Navigation.activelink(NM.activenode parametername)* **do"** or a complex parameter (when the data of the webpage is a set of instances of a concept of the DM) expressed in PRML as: "**When** *Navigation.activelink(NM.activenode* parametername)* **do"**.
- *LoadElement event:* It is associated with the instantiation of a node. The difference with the *navigation event* is that its activation is independent of the link that caused (useful when a set of links have the same target node and we want to personalize during the activation of any of them). In the case of loading a node a parameter is passed containing the information of the root concept (class) of the node loaded. In the case of loading a collection of links the parameter passed would be the set of links.

[2] All the parameters have the prefix NM to indicate that they come from the Navigation Model.

- *Start event* It is associated with the entrance of the user in the website (i.e. the start of the browsing session). In PRML this event is expressed as "**When** *SessionStart* **do**".
- *End event:* It is associated with the exit of the user from the website (i.e. expiration of the browsing session). In PRML this event is expressed as "**When** *SessionEnd* **do**".

The actions we can perform depend on the PRML conformance level supported by the methodology. OO-H supports (fully) both PRML levels. We are not going into detail into the operations supported because we focus here on the behaviour definition. In next section we explain how we have extended PRML for modelling (complex) behaviour aware websites.

4. Extending PRML for Modeling Behaviour Aware Websites

PRML already supports personalization on basis of (simple) user's behaviour (i.e. user's click on a link), however, as already mentioned, we think is important of being able to support the detection of more complex actions of the user. We explain how to do so in subsection 4.1. Moreover PRML supports the definition and (runtime) recognition of (predefined) navigational patterns, being able to personalize in basis of this information. When doing so, we have to face some problems like how do we track the user behaviour and how do we recognize certain behaviour pattern. We are going to explain this subsection 4.2.

5.1 Support for Complex Browsing Behaviour

To support not so trivial browsing behaviour of the user (i.e. user clicks on a link) we need to consider more complex events, like a sequence of clicks on several links in a specific order, etc. For this purpose we have extended PRML events adding the following composite events[3]:

- *NavigationSet event:* It is caused by the activation of a concrete set of navigational links. We can express that all the events should be triggered using the operator "," or express that at least one of the specified events has to be triggered using the "||" operator. We can also state that the events should be triggered in the order they are specified using the "&" operator. In PRML is expressed as: **"When** NavigationSet*[link1(NM.activenode par1),…linkn(NM.activenode parn)]* **do"**. If we want to consider that a certain link has to be visited a concrete number of times and/or a certain number of seconds we can specify it besides the link ID in this way: "**When** NavigationSet[*link1(NM.activenode par1,numClicks,numSec), … linkn(NM.activenode parn, numClicks,numSec)]* **do"**.

[3] Note that the parameters of each of the events are the same as described in section 3

- *LoadSet event:* It is associated with the instantiation of a set of nodes. In PRML is expressed as: "**When** LoadSet*[node1(NM.activenode par1), …noden(NM.activenode parn)]* **do"**. To specify the user has to be browsing a node a certain number of seconds for the event to be triggered and/or the number of times the node has been loaded by the user we express it in PRML as follows: "**When** LoadSet *[node1(NM.activenode par1,numLoads,numSec),… noden(NM.activenode parn,numLoads,numSec)]* **do"**. We use the same set of operators than in the *NavigationSet* event (",", "&", "||") to express order between events and if it is mandatory that all the events should be triggered or not.

Moreover combinations of these two types of events (and also combinations with simple events) can be done in the same rule using the operators "," to express that all the events should be triggered, "&" to express all the events should be triggered in a specific order and "||" to express at least one of the specified events has to be triggered (e.g. "**When** (LoadSet[node1*(NM.activenode par1)* &node2*(NM.activenode par2)*] || NavigationSet[link1*(NM.activenode par1)* ||link2*(NM.activenode par2)*])**do**).

Simple events defined in previous section are also extended in this way:

- *Navigation event*, we can specify the number of seconds the user was browsing in the node resulting from the navigation, and the number of times the link has been activated, in the same way as the *NavigationSet* event.
- *LoadElement event*, we can specify the number of times the node (element) has been loaded by the user and the number of seconds that the user was browsing on it.

We are going to show the definition of (complex) user behaviour actions by means of examples in the case study. For this purpose, let's consider now the (already specified) personalization requirement:

- *Users will see the categories (for messages) sorted by (the user) interest on them. Sort the categories by the ones in which the user has most interest on.*

Two things are required here, first is to detect which are the categories in which the user is most interested on, and second would be sorting the categories by this value.

To detect the interest of the user in the categories we need to track the user behaviour and it is needed to define when we consider that the user has interest on a certain category. This can be defined in several ways, we show here three possible alternatives:

1. *The user has interest on a category when s/he has checked one message of that category.*

In this case, we are going to update the interest degree on a category when the user accesses it. To store/update the value of the *Interest.degree* attribute from the UM we need the following acquisition rule:

```
When Navigation.ViewMessages(NM.Category cat) do
   Foreach i in (UM.User.uToInterest) do
```

```
        If(cat.name=i.ItoCategory.name) then
                SetContent(i.degree,i.degree+10)
        endIf
   endForeach
endWhen
```

This rule is triggered when the user activates the link *ViewMessage*s (see Figure 3). It updates the degree of interest in the category of the consulted message using the *SetContent* statement, in which we specify the attribute to be modified, and the value or formula that calculates the new value. This rule compares each of the instances of the class interest with the consulted instance (to properly update the value). As explained before, in some cases we need to access the data from the navigation prior to trigger the event for using it in rules. For this purpose the rule event carries a parameter containing the visited instance of the Category class (from the NM). This parameter has the prefix "NM". The rest of the data of this rule have the "UM" prefix, indicating that information is stored in the UM data space.

2. *The user has interest on a category when s/he has checked at least 3 messages of that category and has been navigating at least 1 minute on each of the messages pages.*

In this case, we are going to update the interest degree on a category when the user accesses 3 of the messages attached to that category and stays at least 1 minute checking each of the three messages. In this case to store/update the value of the *Interest.degree* attribute from the UM we need the following acquisition rule:

```
When NavigationSet[ViewMessages(NM.Category cat, 3,60)]do
   Foreach i in (UM.User.uToInterest) do
        If(cat.name=i.ItoCategory.name) then
                SetContent(i.degree,i.degree+10)
        endIf
   endForeach
endWhen
```

This rule is triggered when the user activates the link *ViewMessage*s three times and stays 60 seconds at least in each of the message visited. It updates the degree of interest in the category of the consulted message using the *SetContent* statement, in the same way as the previous rule.

3. *The user has interest on a category when s/he has checked at least 3 messages of that category and has been navigating at least 1 minute on that page or if the user posted a comment on one of these messages.*

A new alternative is added for triggering the acquisition rule to update the interest degree on a category: the rule will be also triggered when the user adds a comment on a message visited. In this case we need the following acquisition rule:

```
When NavigationSet[ViewMessages(NM.Category cat ,3,60) ||
AddComment(NM.Message msg)] do
   Foreach i in (UM.User.uToInterest) do
        If(cat.name=i.ItoCategory.name) then
```

```
                SetContent(i.degree,i.degree+10)
        endIf
    endForeach
endWhen
```

The event of this rule gets more complex, this rule is triggered when the user activates the link *ViewMessage*s three times and navigates over the messages at least 60 seconds or when the user clicks on the *AddComment* link operation to add a new comment on a visited message. Once it is triggered it updates the degree of interest in the category of the consulted message in the same way as previous rules.

Once the needed information is stored (*user interest degree*) we need to sort the categories (basing it on the UM data). For this purpose we need a personalization rule:

```
When LoadElement.Category(NM.Category* categories) do
        SortLinks categories orderBy ASC UM.User.uToInterest.degree
endWhen
```

The rule is triggered when the node *Category* is loaded (through any link). It sorts the categories ordered by their user interest (stored in the previously presented rule) in an ascending way. This rule has no condition so when the node is loaded, categories titles are sorted. Note that here we do not specify a loop for sorting the books since *SortLinks* runs over all the instances of the categories set passed as a parameter.

4.2 Support for Behaviour Patterns

Different behavioural patterns are possible in the browsing behaviour of the visitors. In order to better accommodate the users, we can analyze their behavioural patterns, and adapt the site accordingly if we recognize a certain pattern (over a significant amount of time). The definition of such patterns makes easier the tracking of complex user browsing behaviour.
To support behaviour patterns definition and recognition in PRML we add a new type of rules, called *behavioural rules.* These rules track the user browsing behaviour and detect (at runtime) navigational patterns (defined also with behavioural rules). Moreover when a pattern is detected the proper action is performed. The main difference with the rest of PRML rules (i.e. acquisition and personalization rules) is the parameters passed in the events. In this kind of rules we pass (as a parameter) the navigational path that the user is following. We need to define (for each defined pattern) what we consider a navigational path. We show next an example for a browsing behaviour patterns defined in [3].

Direct path pattern:

Intent: provide direct navigation access to information relevant for the particular user, instead of forcing the user to follow each time the same set of navigation tracks.

Solution: adapt the navigation access point of the user if a direct path from that access point to a certain (other) node is detected in a certain

percentage of session (for example, 80%) or in a significant number of times. In that case, a link to the relevant information is added.
Consequence: reduces amount of clicks to relevant information (for the particular user)

When a user enters in the system we start tracking his/her behaviour, keeping the navigational paths that s/he follows. We (only) consider a (finished) navigational path (for this browsing pattern) a path that starts in the page where the user entered the application and finishes in the page in which s/he stays at least 2 minutes. We can see a representation of the navigational path for this pattern in the statechart of figure 4.

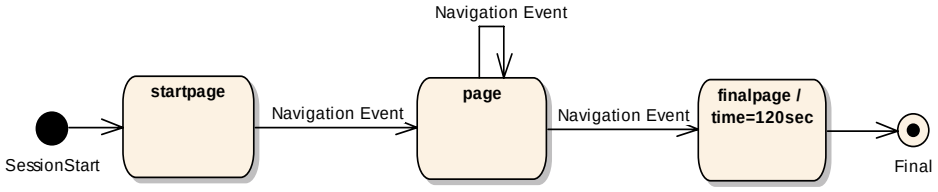

Figure 4: Navigational Path for the Direct path pattern

The navigational paths are stored in the UM, each of them having an identifier. A behavioural pattern can be defined as a certain sequence of navigation actions. We define the following PRML rules to detect the direct path pattern:

```
When SessionStart do
SetContent(UM.UserToDirectPath.path, startpage)
endWhen
```

This rule is triggered when the user enters the website and it initializes the navigational path of the user with the starting page (i.e. the page in which the user starts the session on the website). Next we need to build this navigational path meanwhile the user is browsing the web application. For this purpose we have the following *behavioural* rule:

```
When Navigation.Link(NM.CurrentPath path) do
Foreach a in (UM.UserToDirectPath) do
If (a.path=path) then
        SetContent(a.path, a.path &link)
endIf
endWhen
```

This behavioural rule is triggered when the user activates any link (represented by the *Link* id). As a parameter (as already stated) we have the current path of the user in the website. The path stored in the user model is updated, adding the new browsed link. What is left now is to detect when the path has been “finished”. As aforementioned we consider a navigational path being finished when the user browses a page for at least 2 minutes. We express this with the following *behavioural* rule:

```
When Navigation.Link(NM.CurrentPath path ,1,120) do
```

```
Foreach a in (UM.UserToDirectPath) do
       If (a.path=path) then
              SetContent(a.path, a.path &link &finished)
              SetContent(a.clicks, a.clicks+1)
       endIf
endForeach
endWhen
```

This behavioural rule is also triggered by any link activation (represented by the *Link* ID). In this case we add to the path (list) variable of the user model the last link visited and the keyword *finished*, knowing it is an ended path. We need now a rule to update the clicks on the finished path when the user goes through it:

```
When Navigation.Link(NM.CurrentPath path) do
Foreach a in (UM.UserToDirectPath) do
       If (a.path=path and last(a.path)=finished) then
              SetContent(a.clicks, a.clicks+1)
       endIf
endForeach
endWhen
```

This rule checks if the current visited path has been already detected as a finished one (checking the last element of the list in which the path is stored) and if so the number of clicks on it is increased.
What is left now is to check if the direct path pattern is detected and perform the proper action. We consider that if a user has more than 100 clicks on a navigational path the direct path is detected, and what we do is to add in the home page a shortcut to that path.

```
When SessionStart do
Foreach a in (UM.UserToDirectPath) do
       If (a.clicks>100) then
              Add(shortcut(a.path),homepage)
       endIf
endForeach
endWhen
```

5 Execution Architecture

In OO-H we have created a prototype software application called OPWAC (OO-H Personalized Web Applications Creator) with which we give credit to what we have seen in the previous sections. This tool generates from the OO-H models (the four previously described views), the final (personalized) web pages. Moreover it allows evaluating and performing the set of PRML rules attached to the OO-H models to personalize the final web sites. In figure 5 we show the architecture that follows the OPWAC prototype.

Our main goal has been to generate (personalized) web applications from OO-H based models. We have a web server that interacts with the user, gathering the requests and giving back the response. This website engine generates (on demand) the web pages dynamically from the models (represented in XMI[13] format),

capturing the events produced by the user actions; these events are sent to the PRML evaluator module responsible of evaluating and performing the personalization rules attached to the events that cause the modification of the OO-H diagrams. After these modifications the web engine properly generates the new pages.

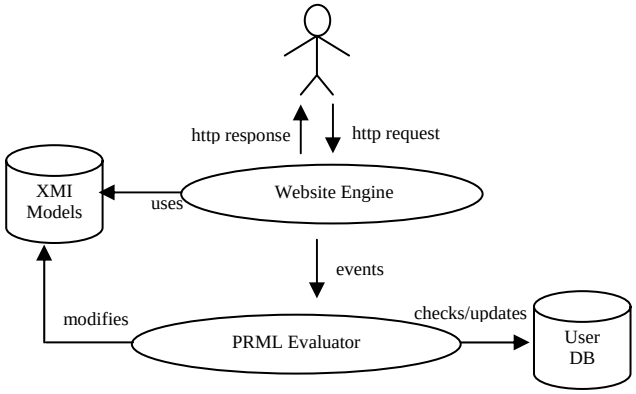

Figure 5. OPWAC architecture

Next we describe in detail the technologies used for implementing this architecture.

5.1 Website Engine

The Website Engine, as we previously said, generates the final (personalized) web pages from the OO-H models. These models are the input of our tool and are represented by means of XML elements (in XMI [13] format). The reason for choosing an XMI representation of the models is that this format can be easily generated from UML models[4]. The only problem is that the OO-H diagram is not UML compliant, but it can be represented by the UML modeling language. To transform the OO-H navigation diagram (NAD) to UML we have extended the syntax and semantics of the UML 2.0 modelling language to express the specific concepts of the NAD, thanks to the UML profiles. To define these profiles UML 2.0 uses stereotypes, which would represent the NAD elements that we want to include in the profile.

To read and process the OO-H models (specified in XMI) for the generation of the final web pages we have used the .NET technology. This technology provides us with the DOM class (XML Document Object Model), with which we can represent in memory the XML documents.

5.2 PRML Evaluator

This module is the responsible for analyzing the events that the Website engine sends for the execution of the personalization rules that will perform modifications

[4] Most UML tools allow this transformation

in the OO-H models. In the NAD components we have added information (thanks to the UML profile created) about the PRML rules to execute (which are stored in a separated file). When a rule is triggered, to evaluate the rule conditions and perform the proper actions we have implemented a .NET component using the ANTLR Parser generator [1]. From the PRML grammar we can generate the syntactic trees which help us to evaluate the rule conditions and perform them if necessary. Finally to execute the actions of the rules, we have implemented in C# the different actions types that we can find in PRML. These actions will modify the XMI models to generate the new web pages from them.

6 Conclusions and future work

In this paper we have presented an approach to model behaviour aware web applications. We use a method independent language called PRML (Personalization Rules Modeling Language) to specify behaviour aware websites. PRML evolved out the experience of OO-H and was designed to be a generic personalization specification method that can be reused for different web design approaches. PRML has been extended to support personalization when a complex behaviour event is triggered (i.e. a sequence of links) and also allows the definition and recognition (at runtime) of user behaviour patterns. We can define such a pattern once and reuse it for its recognition in several websites.

References

1. ANTLR, *ANother Tool for Language Recognition*, http://www.antlr.org/
2. Casteleyn, S., De Troyer, O., Brockmans, S.: *Design Time Support for Adaptive Behaviour in Web Sites*, In Proc. of the 18th ACM Symposium on Applied Computing, Melbourne, USA (2003), pp. 1222 – 1228.
3. Casteleyn, S., Garrigós, I., Plessers, P.: *Pattern Definition to Refine Navigation Structure in Hypermedia/Web Applications* , In Proceedings of the IADIS International Conference WWW/Internet 2004 (ICWI2004), Volume II, pp. 1199-1203, Eds. Isaias, P., Karmakar, N., Publ. IADIS PRESS, ISBN 972-99353-0-0, Madrid, Spain (2004)
4. Dayal U.: *Active Database Management Systems*, In Proc. 3rd Int. Conf on Data and Knowledge Bases, pp 150–169, 1988.
5. De Troyer, O., Casteleyn, S.: *Designing Localized Web Sites*, In Proceedings of the WISE 2004 Conference pp. 547 - 558. Springer-Verlag, Brisbane, Australia (2004)
6. Facca F. M., Ceri S., Armani J. and Demaldé V., *Building Reactive Web Applications.* Poster at WWW2005, Chiba, Japan (2005).
7. Garrigós I., Gómez J., Barna P., Houben G.J.: *A Reusable Personalization Model in Web Application Design.* International Workshop on Web Information Systems Modeling (WISM 2005) July 2005 Sydney, Australia.
8. Gómez, J., Cachero, C., and Pastor, O.: *Conceptual Modelling of Device-Independent Web Applications*, IEEE Multimedia Special Issue on WE (2001) pp 26-39.

9. Houben, G.J., Frasincar, F., Barna, P, and Vdovjak, R.: *Modeling User Input and Hypermedia Dynamics in Hera* International Conference on Web Engineering (ICWE 2004), LNCS 3140, Springer-Verlag, Munich(2004) pp 60-73.
10. Hubert Baumeister, Alexander Knapp, Nora Koch and Gefei Zhang. *Modelling Adaptivity with Aspects*. International Conference on Web Engineering (ICWE 2005), Sydney, Australia, LNCS 3579, Springer Verlag, 406-416, July 2005.
11. Nielsen, J.:*Finding usability problems through heuristic evaluation*.. In Proc. of the SIGCHI Conf on Human factors in computing systems.Monterey, California, United States pp: 373 – 380. 1992.
12. Rossi, G., Schwabe, D. and Guimaraes M. *Designing Personalized Web Applications*. In Proc. of the World Wide Web Conference (WWW'10). Hong Kong May 2001.
13. XML Metadata Interchange www.omg.org/technology/documents/formal/xmi.htm
14. Wu, H., Houben, G.J., De Bra, P.:*AHAM: A Reference Model to Support Adaptive Hypermedia Authoring*, In Proc. of the "Zesde Interdisciplinaire Conferentie Informatiewetenschap", pp. 77-88, Antwerp, 1998.