Modeling Web Applications reacting to User Behaviors

Stefano Ceri, Florian Daniel, Federico M. Facca [*]

Dipartimento di Elettronica e Informazione, Politecnico di Milano
P.zza Leonardo da Vinci 32, I-20133 Milano, Italy

Abstract

Many current research efforts address the problem of personalizing the Web experience for each user with respect to user's identity and/or context. In this paper we propose a new high-level model for the specification of Web applications that takes into account the manner in which users interact with the application for supplying appropriate contents or gathering profile data. We therefore consider entire *behaviors* (rather than single *properties*) as the smallest information units, allowing for automatic restructuring of application components. For this purpose, a high-level *Event-Condition-Action* (ECA) paradigm is proposed, which enables capturing arbitrary (and timed) clicking behaviors. Also, the architecture and components of a first prototype implementation are discussed.

Key words: Adaptive Hypermedia, User Behavior Modeling, WebML, WBM, Web Modeling, Reactivity on the Web

1 Introduction

Today's Internet users are more and more faced with complex Web applications, dynamically generated contents and highly variable site structures. They are continuously confronted with huge amounts of (sometimes irrelevant) contents and changed interaction paths. As a consequence, users may feel uncomfortable when navigating the Web.

[*] Corresponding author.
Email addresses: `ceri@elet.polimi.it` (Stefano Ceri), `daniel@elet.polimi.it` (Florian Daniel), `facca@elet.polimi.it` (Federico M. Facca).

Several techniques aim at augmenting the efficiency of navigation and content delivery. Content *personalization*, for example, allows for more efficiently tailoring of contents to their recipients by taking into account predefined roles or proper user profiles. *Context-aware* or *adaptive* Web applications [1,2] aim at personalizing delivered contents or layout and presentation properties not only with respect to the identity of users, but also by taking into account the context of the interaction involving users and applications. Along a somewhat orthogonal dimension, *workflow-driven* Web applications [3] address the problem of showing the right information at the right time by explicitly modeling the hidden (business) process structure underlying determined usage scenarios, especially within business-oriented domains. Eventually, *usability studies* and *Web log analyses* [4] investigate the usability and thus ergonomics problem by means of an ex-post approach with the purpose of deriving structural weaknesses, checking assumptions made about expected user navigations and mine unforeseen navigation behaviors for already deployed Web applications.

In this paper, we sum up and further extend our previous research on adaptive Web applications [1,5,6] and propose a new approach to (coarse-grained) application adaptation. More precisely, we combine adaptive and process-centric perspectives to design *behavior-aware* Web applications, which allow performing actions in response to the user's fulfillment of predefined navigation patterns. These are described by means of WBM (*Web Behavior Model*), a simple and intuitive model for specifying navigation goals, and allow creating high-level *Event-Condition-Action* rules for expressing novel adaptation requirements. Our proposal adopts WebML for hypertext [7] and adaptation [1] design, but the proposed approach is of general validity and can thus be applied to arbitrary Web applications.

This paper is organized as follows: Section 2 summarizes WebML and its adaptation primitives and describes a small e-learning application used as an example scenario throughout this paper. In Section 3 we introduce WBM, and in Section 4 we combine WebML and WBM for defining proper ECA rules. Section 5 illustrates an applicative example, and Section 6 describes a prototype architecture and discusses experiences gained so far. In Section 7 we provide an overview on related research work and, finally, in Section 8 we address future research efforts and draw some conclusions.

2 WebML and Adaptation

WebML (Web Modeling Language) is a conceptual model for the design of Web applications [7], supported by a proper CASE tool [8]. The WebML method fosters a strong separation of concerns and allows separating the information content from its composition into pages, navigation, and presentation, which

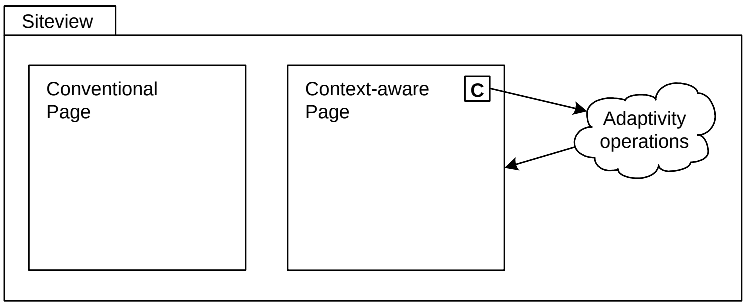

Fig. 1. Example WebML hypertext schema containing a conventional, non-adaptive page and a context-aware one with associated adaptivity operations.

can be defined and evolved independently. Also, primitives for specifying data manipulation operations for updating the site content or interacting with arbitrary external services are provided.

The modeling language offers a set of visual primitives for defining structural schemas that represent the organization and navigation of hypertext interfaces on top of the application data, while for specifying the organization of data the well known Entity-Relationship model is adopted. All visual primitives are accompanied by an XML-based, textual representation, which allows specifying additional detailed properties, not conveniently expressible in the visual notation, and provides the starting point for the automatic generation of the application code.

Recently, WebML has been extended to support the design of context-aware or adaptive Web applications [1]. Adaptivity is associated only to some pages of the application and occurs by refreshing those pages either periodically or when demanded by context conditions. Context data is maintained in a so-called context model within the application's data source. Adaptivity itself may mainly involve (i) contents published by specific pages, (ii) automatic navigation actions toward other pages of the hypertext, or (iii) adaptation of the overall hypertext structure (i.e., in a multi-channel environment).

In order to achieve these effects, chains of operations are associated to adaptive pages and express the actual actions to be carried out upon context-triggered page refreshes. Figure 1 graphically summarizes the described scenario: there is one site view containing one conventional page and one context-aware page (labeled with a `C`) together with its associated set of adaptivity operations. The directed arcs (links) serve a twofold purpose in this particular configuration: on the one hand they allow associating adaptivity operations to their pages, while on the other hand they provide a mechanism for passing parameters among the two computationally independent fragments of hypertext. Adaptivity operations are evaluated only for `C`-labeled pages and only after their automatic refresh, leading to the above mentioned adaptivity effects.

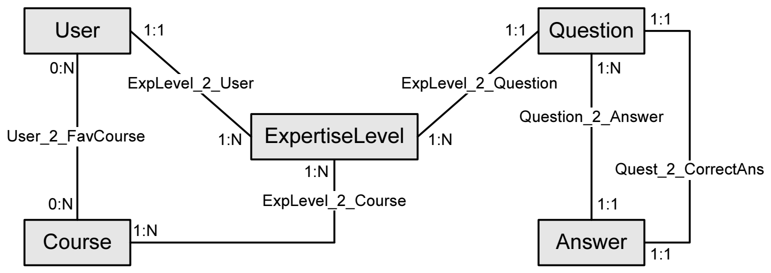

Fig. 2. An example of Entity-Relationship schema for an e-learning application.

In this paper we will not adopt any context model within the application's data source for managing adaptivity as proposed in [1]. Instead, we will trigger adaptivity according to user behaviors and, thus, by monitoring the execution of WBM specifications (see Section 3). For further details on WebML and its adaptivity extension, the reader is referred to [7] and [1].

2.1 Example Scenario

Throughout this paper we will make use of design examples to explain the novel concepts to be introduced. This section provides the adopted e-learning reference scenario by means of a WebML modeling example.

The non-adaptive application allows users to browse courses of his/her personal expertise level and to test the acquired knowledge by answering related questions, thus possibly enhancing the associated expertise level. Figure 2 depicts the E/R schema underlying the e-learning application: each user has a set of favorite courses and an associated level of expertise. Each course is related to one or more levels of expertise. For each level of expertise there is a set of questions; each question is associated to a set of possible answers and to one correct answer. To simplify the scenario, test questions are related only to a specific expertise level and do not depend on courses.

The WebML hypertext model of the e-learning application is depicted in Figure 3. The *Home* page contains *User Data* and a list of *Suggested Courses*. The *Get Unit* allows accessing the user's identifier, while the selector conditions below the units allow binding a unit to a data entity and personalizing the displayed items by applying filter conditions. From the *Home* page the user can ask for the *Courses* page. When requesting that page, the user's possible level of expertise is forwarded to an `If` operation unit, which checks whether there already exists an expertise level or not. If the user already has an associated level of expertise, he/she is forwarded to the *Courses* page (`KO` link); otherwise, the user is provided with the *Test* page (`OK` link). In this case, a multiple choice test for the lowest level of knowledge is proposed to the user. The *Questions* multidata unit presents a selection of questions according to the

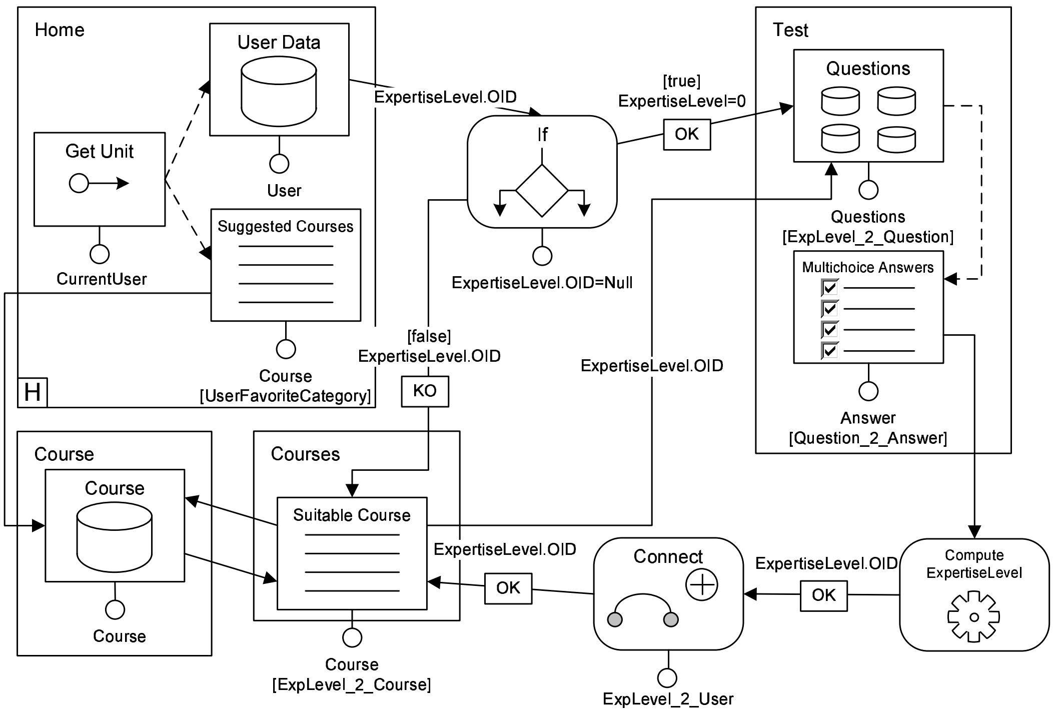

Fig. 3. The WebML model of the proposed educational Web site.

`ExpertiseLevel` parameter passed by the incoming `OK` link. According to the selected question a corresponding *Answer* multichoice index unit is proposed to the user. When submitting the filled test, an *ExpertiseLevel* is computed and associated to the user by the *Connect* operation unit. Finally, the user is redirected to the *Courses* page, where suitable concepts are presented. From here, the user can browse new contents (*Course* page) or navigate to the *Test* page and perform a new test to verify if his level is increased after having studied new contents.

3 The Web Behavior Model

The *Web Behavior Model* (WBM) is a timed state-transition automaton for representing classes of user behaviors on the Web. WBM does not serve for deriving runtime navigation behaviors, but instead allows describing navigation patterns (at design time) without requiring a profound knowledge of the actual application structure.

Graphically, WBM models are expressed as labeled graphs, allowing for an easily comprehensible syntax (cf. Figure 4). A *state* represents the user's inspection of a specific portion of hypertext (i.e., a page or a collection of pages). State labels are mandatory and correspond to names of pages or page collections. A *transition* represents a navigation from one such portion to another

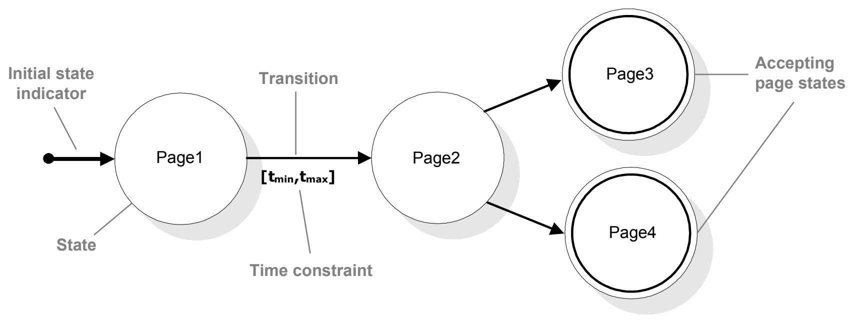

Fig. 4. Example of WBM script with state, link, time constraints and multiple exiting transitions from one state.

and, thus, the evolving from one state to another. Each WBM specification, called *script*, has at least an initial state, indicated by an incoming unlabeled arc, and at least one accepting state, highlighted by double border lines. Initial states cannot also be accepting states. Each transition from a source to a target state may be labeled with a pair $[t_{min}, t_{max}]$ expressing a time interval within which the transition can occur. Either t_{min} or t_{max} may be missing, indicating open interval boundaries. If a transition does not fire within t_{max} time units, it can no longer occur; on the other hand, navigation actions that occur before t_{min} are lost.

One important aspect of WBM models is that not all navigation alternatives must be covered. As the aim of WBM is to capture a concise set of user interactions, describing particular navigation goals and respective “milestones”, only a subset of all possible navigation alternatives is relevant. E-commerce Web sites, for example, make heavy use of so-called *access-pages* that only serve the purpose of providing users with browsable categories for retrieving the actual products offered. Furthermore, Web sites usually provide several different access paths toward their core contents. Therefore, by concentrating only on those interactions that really express navigation goals, WBM allows both abstracting from unnecessary details and defining small and easily comprehensible specifications. Only performing specified target interactions – in the modeled order – may thus cause WBM transitions.

Figure 4 shows an example WBM script. The initial state corresponds to *Page1*. The transition from the first state to the second state occurs only if the user requests *Page2* within t_{min} and t_{max} time units from the moment the script has been initiated, otherwise the script ignores the navigation and remains in its current state. The script in Figure 4 also shows two transitions exiting from state *Page2*. The states labeled *Page4* and *Page3* are “competing”, as a browsing activity in *Page2* may lead to either *Page4* or *Page3*.

In WBM it is possible, and sometimes convenient, to use overlapping states, i.e., states corresponding to overlapping portions of hypertext. If two competing states are overlapping, two transitions may trigger simultaneously.

3.1 WBM Formal model

WBM belongs to the class of timed finite state automata. Starting from this class of automata, as discussed in [9,10], and from the concepts previously introduced, we now give a formal and concise definition of WBM as timed finite state automaton. Step by step we will enrich the formal definition of finite state automata with novel concepts to fully reflect the semantics of WBM.

Definition 1 (Finite State Automaton) *A* finite state automaton *is a tuple $F = (\Sigma, S, S_0, S_F, E)$, where: Σ is a finite set of input symbols; S is a finite, nonempty set of states; $S_0 \subseteq S$ is a nonempty set of starting states; $S_F \subseteq S$ is a nonempty set of final states, such that $S_0 \cap S_F = \emptyset$; $E \subseteq S \times S \times \Sigma$ is a set of* transitions *or* edges*.*

A timed finite state automaton can be defined as a finite state automaton with transitions constrained in time. Thus, Definition 1 needs to be extended with a clock and the relative definition of time constraints. In this paper we use a discrete-time model, hence the set of nonnegative integer numbers, $\mathbb{N}$, is chosen as the time domain.

Definition 2 (Time Sequence) *A* time sequence $\tau = \tau_1\tau_2\ldots$ *is an infinite sequence of time values $\tau_i \in \mathbb{N}$ with $i \geq 1$, satisfying the following constraints:*

(1) Monotonicity: $\tau_i < \tau_{i+1}$ *for all* $i \geq 1$.
(2) Progress: *For every $t \in \mathbb{N}$ there is some $i \geq 1$ such that $\tau_i > t$.*

A timed word Σ_t *over an input set of symbols Σ is a pair Σ_t=(σ, τ) where $\sigma = \sigma_1\sigma_2\ldots$ is an infinite word over Σ and τ is a time sequence.*

If a timed word Σ_t=(σ, τ) is used as input to an automaton, then the time τ_i represents the time of the occurrence of the symbol σ_i.

Now, according to [9], we extend the finite state automaton to interpret timed words and associate a clock to the automaton. A *clock* is a variable with values in $\mathbb{N}$. Given that all time values are relative to state transitions, for simplicity in our automaton the clock is reset to zero at each state-transition. The absolute values of time events can be computed by summing the absolute time of the last state transition to the relative time of the given event. Before finally defining the automaton we also need to formalize time constraints over transitions:

Definition 3 (Time Constraint) *Given a clock x, a* time constraint *δ is defined inductively by $\delta := x \geq c || x \leq c || c_1 \leq x \leq c_2$, where $c, c_1, c_2 \in \mathbb{Q}_+$ and $c_1 < c_2$.*

According to the previous definitions, timed finite state automata can thus be defined as follows:

Definition 4 (Timed Finite State Automaton) *A* timed finite state automaton *is a tuple $F = (\Sigma_t, S, S_0, S_F, E, x)$, where: Σ_t is a timed word of input symbols; S,S_0,S_F are defined in Definition 1; x is the automaton clock; $E \subseteq S \times S \times \Sigma_t \times \Phi$ is the set of transitions, with Φ being a set of time constraints δ over the clock x. An edge (s, s', σ, δ) represents a transition from state s to state s' on input $\sigma \in \Sigma$ and subject to the clock constraint δ over x.*

In order to describe user behaviors within a Web application by means of a timed finite state automaton, we now introduce a minimal and generic definition of hypertext.

Definition 5 (Hypertext) *A* hypertext *is a couple $H = (P, L)$ where P is a set of pages and $L \subseteq P \times P$ is a set of links, such that every link in L connects two and only two pages in P.*

Given the definitions above, we can now define the Web Behavior Model (WBM).

Definition 6 (Web Behavior Model) *Given a hypertext $H = (P, L)$ and an timed finite state automaton $F = (\Sigma_t, S, S_0, S_F, E, x)$, a* Web Behavior Model *is a couple $WBM(H, F) = (P_l, L_l)$ where:*

- *$P_l : S \rightarrow P \cup P_c \cup \{*\} \cup WBM_i(H, F_i)$ is a function that maps any state $s \in S$ to a page $p \in P$ of the hypertext H, or to a collection of pages $P_c \subseteq \mathcal{P}(P)$, or to the special symbol $*$ denoting any page of the hypertext H, or to any other Web Behavior Model $WBM_i(H, F_i)$ defined over the hypertext H.*
- *$L_l : T \rightarrow L \cup \{*\}$ is a function that maps any transaction $e \in E$ to a link of the hypertext H, or to the special symbol $*$ denoting an unconstrained navigation between the pages corresponding to the two states connected by the transition e.*

This formal definition of WBM is based on the assumption that the automaton does not terminate after receiving unexpected input symbols, and instead keeps its current state, while waiting for a valid input symbol without resetting its clock. Also, as the attentive reader may have noticed, WBM models can be described recursively, which allows complex behaviors to be represented by means of sub-models.

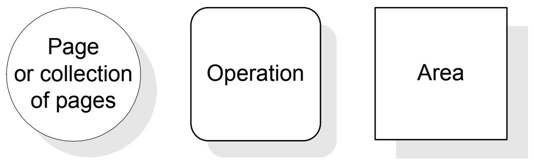

Fig. 5. WBM extended set of symbols.

3.2 WBM and WebML

In this paper, we combine WBM and WebML for modeling Web applications. Exploiting some structural peculiarities of WebML-based applications allows us to further refine WBM transition constraints as introduced in the previous section. In addition to time constraints, in this section we further define so-called *state* and *link constraints* for augmenting the expressive power of WBM models.

WebML hypertext schemas are based on three core elements: areas, pages and units. As shown in Figure 5, taking them into account by means of new WBM state symbols allows for easier and more expressive model definitions. Furthermore, contents are published by so-called content units bound to data entities, which can be "queried" to retrieve details about the navigated contents. For this purpose, we now introduce WBM variables, assignments and predicates.

Variables are untyped and named by alphanumerical character sequences, beginning with a character. *Assignments* are formulas `<variable name> := <term>`, where a `<term>` is an arithmetic expression using either constant values, functions, or variables. *Predicates* are arbitrary Boolean expressions `<term> <comp> <term>`, where a `<term>` is an arithmetic expression using either constant values, functions, or variables; `<comp>` is one of the comparators $=, \leq, \geq, \neq, <$ and $>$. Predicates can be compounded to form complex expressions: `<pred> <logicomp> <pred>`, where a `<logicomp>` is one of the two logic operators $\wedge$ (AND), $\vee$ (OR).

To specify *state constraints* over contents and store variable values, four basic *functions* can be used within predicates and assignments:

```
      Display(<Data unit name>, <Attribute name>)
     Selected(<Data unit name>, <Attribute name>)
        Entry(<Entry unit name>, <Form element>)
Parameter(<Operation unit name>, <Attribute name>)
```

The `Display` function applies either to data units, returning the value of an attribute of the displayed entity, or to multi data and index units, returning the set of values of an attribute of the displayed entities. Aggregation functions (such as `SUM, MIN, MAX, AVG, COUNT`) can be applied to sets of values in

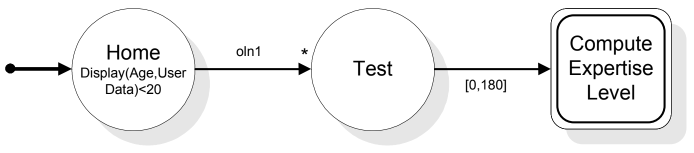

Fig. 6. A WBM script with operation, link constraint and predicates.

order to compute scalar values. The `Selected` function applies to index units only, returning the attribute value of the entity that has been selected by the user. The `Entry` function applies to entry units and returns the value of one form field entered by the user. The `Parameter` function applies to operation units, returning the value of one of the parameters assigned in the operation call.

Referring to the *Course* page of the Web application introduced in Section 2.1, the example assignment

```
x := Display(Course, OID)
```

assigns the *OID* attribute of the item being displayed by the *Course* data unit to the variable x. Likewise, the predicate

```
Display(Course, Category) = "Web"
```

verifies whether the *Category* attribute of the item being displayed by the *Course* data unit equals the string “Web” or not. A predicate is satisfied whenever the expressed condition evaluates to *true*.

Link constraints are based on WebML link identifiers and allow restricting WBM transitions to explicitly specified links. Link constraints are expressed by labeling transitions with link identifiers. For distinguishing between entering and exiting links, the following policy is adopted: a ‘$*$’ near the begin of a transition arc constrains the link to be an outgoing link; a ‘$*$’ near the end of a transition arc constrains the link to be an incoming link; if a ‘$*$’ is specified on both sides of an arc, the relative link must connect directly the two pages or operations; when the ‘$*$’ is omitted, the transition refers to any path containing the specified link.

The WBM script in Figure 6, based on our reference scenario, illustrates the novel concepts. The depicted script aims at identifying newly registered users that are younger than 20 years and take less than 180 seconds to answer the proposed questions. The script starts when entering the *Home* page, displaying a *User Data* unit with *Age* attribute less than 20. The transition to the second state is enabled only if the user reaches the page *Test* by an incoming *OK* link, we suppose denoted by the WebML identifier *oln1*. Finally, the accepting state is reached when the user requests the *Compute ExpertiseLevel* operation unit within 180 seconds from the activation of the previous state.

Despite the use of WebML for specifying hypertexts in this paper, WBM without its WebML-specific extensions is designed to describe navigation behaviors on top of arbitrary hypertexts. Nevertheless, the use of state and link constraints requires a binding to other models providing Web application structure and contents.

4 Reacting to User Behaviors

In order to be able to react to observed behaviors and to adapt the running application to novel requirements, we now combine WebML and WBM. For this purpose, we transform the context-triggered adaptivity mechanism provided by WebML into a WBM-triggered mechanism. Consequently, adaptivity occurs in reaction to the fulfillment of entire WBM scripts, which can be derived from the user's navigation behavior as outlined in the previous section. According to [1], possible reactions comprise:

- Adaptivity of contents published by specific pages;
- Automatic execution of navigation actions toward other pages;
- Automatic execution of operations of services;
- Adaptivity of the overall hypertext structure.

Although independent from one another, expressing adaptation as a combination of WBM scripts and WebML adaptivity primitives intrinsically leads to a high-level ECA paradigm for specifying adaptivity. Commonly, ECA rules respect the general syntax

`on` *event* `if` *condition* `do` *action*

where the event part specifies *when* the rule should be triggered, the condition part assesses whether given *constraints* are satisfied, and the action part states the *actions* to be automatically performed if the condition holds. When specifying behavior-aware Web applications, the event consists of a *page* or *operation request*, the condition requires the fulfillment of a predefined navigation pattern (expressed as *WBM script*), and the action part specifies some adaptivity actions to be forced on the Web application and expressed as a WebML *operation chain*.

While the progression of activated WBM scripts takes into account all navigations performed by a user, the evaluation of entire rules is restricted to a subset of the application's pages. This subset determines the so-called *scope* of the rule and is specified by labeling adaptive pages with an `A`-label in the WebML hypertext model and explicitly associating those pages to the rule's WBM script. Accordingly, events for one specific rule are generated only when

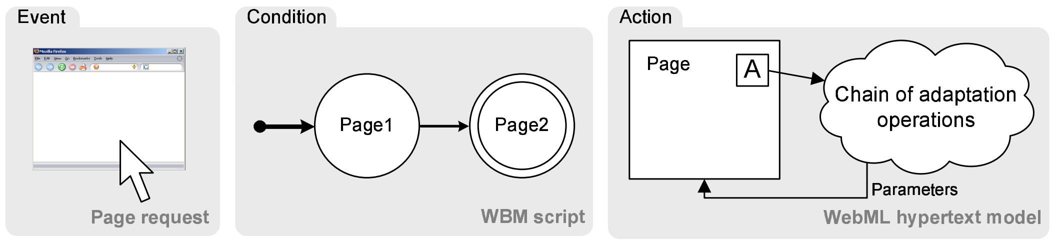

Fig. 7. High-level ECA rule components.

requesting pages within the scope of that rule. As a consequence, also the condition within the rule (WBM script termination) is evaluated only after proper events of the respective rule, and – in case of script termination – the operations indicated by the link exiting the page's `A`-label are executed.

Figure 7 graphically summarizes the outlined rule construct: The rule reacts to a user's visit to *Page1* followed by a visit to *Page2* at some stage after his/her visit to *Page1*. The expressed rule condition thus only holds when the script gets to the accepting state *Page2*. Once the accepting state is reached and the user navigates one of the pages within the rule's scope, the operations associated to that page (abstracted as the cloud in Figure 7) are executed and possible adaptations may be performed.

Pages may have several competing rules associated with them and may thus be part of the scopes of different rules. For resolving possible conflicts among concurrently activated rules (each rule may have different associated action parts), proper rule *priorities* allow selecting the rule with highest priority. Due to the fact that executing the actions of one rule may alter the overall hypertext structure and thus invalidate the semantics of the other simultaneously activated rules, their actions are discarded. Rules are in total priority order, based on explicit numbering or implicit rule creation time.

When considering priorities as properties of rules, rules can be described by the following 4-tuple:

$$\langle \text{Navigation(Scope)}, [\text{WBM Script},]\text{WebML Operations}[, \text{Priority}]\rangle$$

where the Scope represents the extent (pages, areas or site views) within which the rule reacts to navigation events. The *WBM Script* describes the *condition* part of the rule, and the *WebML Operations* specify the *action* part. When the optional *WBM Script* component is omitted, the condition part of the rule always evaluates to `true` and possible adaptation operations are executed at page access. Finally, *Priority* is an optional integer expressing the rule's priority with respect to others.

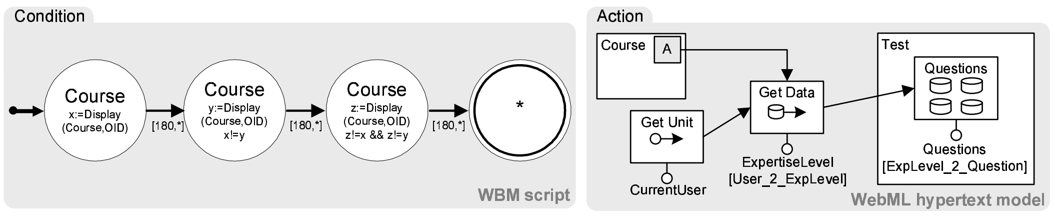

Fig. 8. An ECA rule to trigger the evaluation of a student's knowledge level. The event part (user click) is omitted for simplicity.

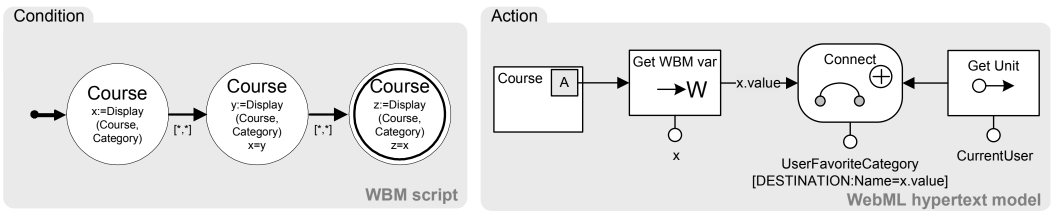
Condition
Course
x:=Display (Course, Category)
[*,*]
Course
y:=Display (Course, Category) x=y
[*,*]
Course
z:=Display (Course, Category) z=x
WBM script
Action
Course
A
Get WBM var
W
x
x.value
Connect
UserFavoriteCategory [DESTINATION:Name=x.value]
Get Unit
CurrentUser
WebML hypertext model

Fig. 9. An ECA rule to profile user preferences.

5 An E-learning Case Study

In the following we describe two examples that add a proper adaptation layer to the Web application presented in Section 2.1.

Example 1. Evolving the Level of User Expertise. Figure 8 models an ECA rule to redirect the user to the *Test* page for the next experience level after having visited 3 courses (i.e., 3 different instances of *Course* pages), spending at least 3 minutes over each page. The $*$ in the final state of the WBM script specifies the acceptance of any arbitrary page. The WebML operation chain for adaptation is actually performed when the user asks again for a *Course* page. When the chain is activated, the expertise level of the current user is retrieved by the `Get Data` unit and used to provide the user with a suitable set of questions for his/her expertise level.

Example 2. User Profiling. Suppose we want to personalize the application according to the user's preferences traceable from his/her navigational choices (cf. Figure 9). The script detects that a user is interested in a certain category of courses, whenever he/she navigates at least three different *Course* pages presenting three courses belonging to the same category. The identified category is stored within the variable x. In response to this behavior, the WebML operation chain stores the derived preference: the value of the variable x is retrieved by the `Get WBM Variable` unit and the identified category is associated to the current user. Now, when the user enters the *Home* page, courses belonging to the same category are automatically presented by means of the *Suggested Courses* unit (cf. Figure 3).

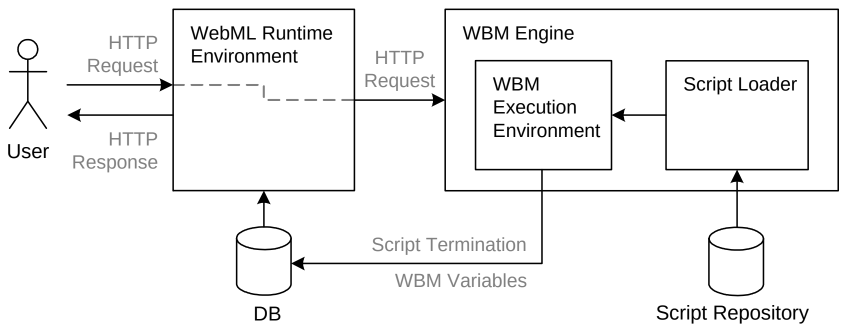

Fig. 10. Functional architecture of the overall behavior-aware system.

6 System Architecture

Executing behavior-aware Web applications – in addition to the standard WebML runtime environment – requires proper runtime support for WBM scripts. As illustrated by Figure 10, this task is addressed by a suitable *WBM Engine*, which manages WBM scripts based on tracked HTTP requests. More precisely, HTTP requests are automatically forwarded to the WBM engine by the WebML runtime environment, which hosts the actual application. Users interact only with the Web application itself and are not aware of the WBM engine behind it.

The WBM engine collects and evaluates tracked, user-generated HTTP requests for (i) instantiating new scripts at runtime, and (ii) enhancing the states of possible running WBM scripts, as well as (iii) communicating possible script terminations. Script instantiation is managed by a proper *Script loader* module based on tracked page requests and the scripts' starting pages (those indicated by their initial states). The set of scripts that can be instantiated for a particular application is retrieved from a *Script Repository.* A dedicated *WBM Execution Environment*, on the other hand, is in charge of progressing instantiated, running scripts. Each user has his/her own set of scripts, which are executed in a completely independent way. Once a script reaches its accepting state, the execution environment communicates the successful termination to the Web application by modifying suitable data structures within the shared database. Those data structures also hold the values of possible WBM variables used during script execution.

After successful termination of a WBM script, the Web application possesses all the necessary data for executing the possibly associated adaptation actions. As soon as a user requests one of the pages within the scope of the high-level rule whose condition is satisfied by the terminated WBM script, the Web application executes the the adaptation operations associated to the requested page. For this purpose, page computation starts by checking whether scripts connected to the page have terminated or not, before proceeding with the actual rendering of the page. If there are terminated scripts for that page, one or more rules could be executed. In case of multiple candidate rules, rule priorities help determining the rule with highest priority. Thus, computation proceeds with the determined adaptation operations, producing effects as described in Section 4. Only afterward and in case of no automatic navigation actions, computation continues with the actual page and a suitable HTTP response is produced.

Synchronous as well as *asynchronous* rule execution models can be implemented. In the synchronous case, when requesting a page within the scope of a rule, the page request is immediately forwarded to the WBM engine and page computation proceeds only after receiving a respective status notification from the WBM engine. In this way, possible WBM scripts are updated before page computation, and possible script terminations can cause an immediate performing of the respective rule's actions. This implies that the speed of page computations depends on the performance of the WBM engine. Although some applications could demand for synchronous configurations, we concentrated on a deferred, asynchronous configuration.

When adopting an asynchronous configuration, page requests are satisfied immediately and forwarded only after page computation. Possible adaptations are performed after a predefined time interval by automatically refreshing the page or at the next page request (if generated before expiration of the refresh interval). This allows for better parallelization of the Web application and the WBM engine, as well as short response times.

6.1 WBM Engine Implementation

The implementation of rule engines for active databases is a well known and studied topic in the literature on database systems. Our problem of handling user sessions and WBM scripts resembles to the problem of handling transactions and rules in active databases. Accordingly, the implementation of the WBM Engine has been inspired by the Starbust Active Database [11].

More precisely, the implementation maintains a catalog of WBM scripts, where each script has associated with it the collection of user sessions that activated it. As the number of instances of user sessions in a heavily used Web application is much greater than the number of defined WBM scripts, it is more efficient to associate user sessions to scripts than running scripts for each single user session. This reduces the effort needed to maintain the data structures in memory and improves the overall system performance. The WBM engine thus inserts tokens, representing single user sessions, into the scripts' runtime data structures. Tokens advance according to script state changes and are removed when entering an accepting state. Removing a token implies communicating the successful termination of the respective script to the database.

However, user sessions are not permanent objects. Due to the HTTP protocol, it is not possible to establish when a user session terminates. This may cause a high number of running WBM scripts never to reach an accepting state. To solve this problem and to handle WBM time constraints, the WBM Engine makes use of a garbage collector, which performs cyclic checks and clean-ups of activated scripts with inactive sessions, where inactivity implies no user navigations for a predefined interval of time. In order to guarantee the consistency of user click streams, tracked page requests are stored in a chronologically ordered queue. The described tracking process thus requires as input the complete URL navigated by the user, the timestamp of the request, an identifier of the user (e.g. the session identifier), as well as the identifier of the Web application itself[1].

6.2 Prototype Experiments

A first prototype of the presented WBM engine has been developed and tested by implementing the behavior-aware Web application outlined in this paper (see Section 2.1). We have fully implemented link and time constraints as well as most of the mechanisms required by WBM state constraints. Results from experiments are quite positive: experiments proved the viability of the approach, and the use of the asynchronous execution model effectively avoids WBM engine response times impacting on user navigation.

Experiments also revealed a performance problem: we observed an excessive lag between the start of a notification of a page request and the final computation of the new state by the WBM engine (around 2.5 seconds to manage 100 user requests). Further studies proved that the bottleneck of the system was not the WBM engine (a stand-alone version of the engine can process the same 100 requests in less than 60 milliseconds, and can efficiently support WBM scripts of greater complexity than the ones described in this paper). Rather, the bottleneck was detected in the generation of SOAP messages by the Web application, as we use Web services for the communication between the Web application and the WBM engine. This performance problem will therefore be fixed in the upcoming prototype, which also will allow testing the impact of complex state constraints[2].

[1] This information could be reconstructed as well from the requested URL.
[2] Experiments were realized using an AMD AthlonXP 1800+, 512MB of RAM and with Tomcat as Web server.

7 Related Work

The ECA rule paradigm was first implemented in active database systems in the early nineties [11,12] to improve the robustness and maintainability of database applications. Recently, they have also been exploited in other contexts such as XML [13], to incorporate reactive functionality in XML documents, and the Semantic Web [14], to allow reactive behavior in ontology evolutions. Our research explicitly adds ECA-rules to Web applications to allow adaptation based on user behaviors.

A number of paradigms able to model a user's interaction with the Web have been proposed [15,16]. Most of these models have been designed more to describe the navigation model of Web applications and less the user interacting with applications. Nevertheless, their semantics can be extended to model user behaviors disregarding the navigation design of the Web application. These models have a strong formal definition, since they are based on well known and established formal models like Petri Nets [15] or UML StateCharts [16].

In the early age of hypertext applications, Stotts et al. [15] introduced *Trellis*, a model based on Petri Nets, to describe hypertext semantics. This research is derived from efforts in the software engineering area to model user interactions [17] and interfaces [18]. Trellis, hence, was not intended to model the semantics of user navigations. Possible user navigation paths caused by interactions with the browser – like the use of the *back button* or the *history mechanism* – are not contemplated by the model. Furthermore, the Petri Nets used by Trellis are not timed. Therefore, time, one of the most relevant features to model user browsing semantics, cannot be captured by this model.

HMBS (Hypertext Model Based on Statecharts) [16] is a navigation-oriented model for hypertext based on UML Statecharts. Like Trellis, HMBS is not intended to describe the browsing semantics from the user's point of view; e.g. transactions are always mapped to existing links of the modelled hypertext. An interesting feature of HMBS is the use of hierarchies, that allow using an HMBS model as a refinement of a state of an HMBS model. Like Trellis, HMBS would need to be extended with a timed semantics to properly capture relevant user navigation behaviors.

WBM, on the other hand, is a general purpose model to describe user interactions at a high level of abstraction, focusing only on navigational alternatives relevant to the user and leaving out irrelevant in-between steps. It includes a timed semantics that permits fully capturing user interactions with the application and makes use of hierarchies that allow describing complex behavioral models by nested simpler models. Moreover, WBM includes predicates, that allow specifying queries over displayed contents in dynamic Web pages, and assignments that permit maintaining displayed contents as state variables and to use them within predicates. Finally, WBM has an visual paradigm that allows designers to easily specify arbitrary user behaviors.

A variety of design models have been proposed for Adaptive Hypermedia systems [19–21]. While most of these methods differ in approach, all methods aim to provide mechanisms for describing Adaptive Hypermedia (see [22,23] for a survey). Most of them do not use a full ECA paradigm and rather deal with a CA paradigm [21]. Others [19] do not use an ECA paradigm and hence do not refer directly to it or do not propose a formal model. Some of them focus only on the adaptation, disregarding an effective description of the user's behavior that should trigger the adaptation [24]. A comprehensive overview of commercially available solutions is presented in [25]. The author points out that commercial user modeling solutions are very behavior-oriented: observed user actions or action patterns often lead directly to adaptations without an explicit representation of the user characteristics.

In the literature, AHAM [19] is often referred to as the reference model for Adaptive Hypertext. It is based on Dexter [26], an early model for hypertext, and uses maps of concepts. AHAM presents many valid ideas (e.g. the 3-layer model capturing all the adaptive semantics) but is more suited to the e-learning domain. Recently AHAM evolved into the AHA! project [27], originating a powerful tool to develop and maintain adaptive Web applications. AHA! makes use of advanced CA rules, based on user preferences and already visited links. The AHA! toolset is mainly based on textual editing for creating rules. Lately the project focused its interest on rule termination and confluence, with the aim of detecting possible loops already at design time.

The model introduced in [24] extends WSDM [28], a Web design method, with an Adaptation Specification Language that allows specifying adaptive behaviors. In particular, the extension allows specifying deep adaptation in the Web application model, but lacks expressive power as regards the specification of the user's behavior that triggers the adaptation. No discussion of an implementation of the proposed design method is provided.

In [21] the authors propose a Software Engineering oriented model based on UML and OCL to describe in a visual and formal way adaptive hypertext applications. The adaptation model is based on a Condition-Action paradigm that allows expressing conditions on the user's behavior. The proposed visual notation perhaps lacks immediacy due to the use of a visual paradigm born outside the Web area.

8 Conclusion and Future Work

In this paper we have proposed a general purpose approach for building behavior-aware Web applications. Our proposal combines WebML and WBM into a visual ECA paradigm that opens the road to the implementation of high-level CASE tools for designing advanced Web sites.

When combined, WebML and WBM yield a very powerful model, with adequate expressive power for capturing highly sophisticated Web dynamics. WBM enables monitoring of behaviors ranging from rather coarse to very detailed event sequences; the binding of WBM predicates to WebML enables the specification of each event in terms of hypertext elements (pages and links) and of the application content. While the combined expressive power is quite strong, we also believe that the two models should be kept distinct, so as to enable the specification of WebML applications (totally independent from WBM) and of WBM scripts (that can be progressively refined and finally bound to WebML concepts).

In our future work, we plan to extend our initial prototype of a reactive Web system to support classical ECA features, including more sophisticated policies for dealing with priorities and conflicts. Currently, we adopt the simple policy of always choosing the rule of highest priority for execution. Furthermore, we did not consider the problems of rule termination yet, which might arise when rules trigger each other. Thanks to the strict relation between WBM and WebML we believe that some termination problem can be detected at design time; but it is neither possible nor useful to constrain rule sets at design time so as to avoid any cyclic behavior, and therefore we will need to support monitoring of nonterminating behaviors at runtime. Dynamic activation and deactivation of rules and of rule groups is already under consideration and the current prototype includes a preliminary version of such features. We are also investigating the use of modeling primitives similar to UML statecharts for expressing WBM; this would allow us to adapt existing case tools for the specification of WBM scripts.

Our first prototype demonstrates the applicability of our approach, as it supports complex rules and therefore can build complex reactive applications. The implementation of a second generation prototype is well under way, offering optimized rule management and full graphic user interfaces to designers.

Acknowledgement
We acknowledge the contribution of Vera Demaldè and Jacopo Armani to the definition of the WBM model and the commitment of Luca Sonzogni and Cristian Rossi to the implementation of the WBM engine and the prototype application.

References

[1] S. Ceri, F. Daniel, M. Matera, Extending WebML for Modeling Multi-Channel Context-Aware Web Applications, in: Proceedings of Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03), Rome, Italy, December 12 -13, 2003, IEEE Press, 2003, pp. 225–233.

[2] A. C. Finkelstein, A. Savigni, G. Kappel, W. Retschitzegger, E. Kimmerstorfer, W. Schwinger, T. Hofer, B. Pröll, C. Feichtner, Ubiquitous Web Application Development - A Framework for Understanding, in: Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI), 2002, pp. 431–438.

[3] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, I. Manolescu, Specification and Design of Workflow-Driven Hypertexts, Journal of Web Engineering (JWE) 1 (2) (2003) 163–182.

[4] F. M. Facca, P. L. Lanzi, Mining interesting knowledge from weblogs: a survey, Data Knowl. Eng. 53 (3) (2005) 225–241.

[5] F. M. Facca, S. Ceri, J. Armani, V. Demaldé, Building reactive web applications, in: Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005 - Special interest tracks and posters, ACM, 2005, pp. 1058–1059.

[6] S. Ceri, F. Daniel, V. Demaldé, F. M. Facca, An Approach to User-Behavior-Aware Web Applications, in: Web Engineering, 5th International Conference, ICWE 2005, Sydney, Australia, July 27-29, 2005, Proceedings, Vol. 3579 of Lecture Notes in Computer Science, Springer, 2005, pp. 417–428.

[7] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera, Designing Data-Intensive Web Applications, Morgan Kauffmann, 2002.

[8] WebModels srl: WebRatio. `http://www.webratio.com`.

[9] R. Alur, D. L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (2) (1994) 183–235.

[10] T. A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine, Symbolic model checking for real-time systems, Inf. Comput. 111 (2) (1994) 193–244.

[11] J. Widom, The Starburst Active Database Rule System, IEEE Trans. Knowl. Data Eng. 8 (4) (1996) 583–595.

[12] S. Ceri, P. Fraternali, S. Paraboschi, L. Tanca, Active Rule Management in Chimera, in: Active Database Systems: Triggers and Rules for Advanced Database Processing, Morgan Kaufmann, San Francisco, California, 1996.

[13] A. Bonifati, S. Ceri, S. Paraboschi, Active rules for XML: A new paradigm for E-services, The VLDB Journal 10 (1) (2001) 39–47.

[14] G. Papamarkos, A. Poulovassilis, P. T. Wood, Event-Condition-Action Rule Languages for the Semantic Web, in: Proceedings of SWDB'03, Berlin, Germany, September 7-8, 2003, 2003, pp. 309–327.

[15] P. D. Stotts, R. Furuta, Petri-Net-Based Hypertext: Document Structure with Browsing Semantics, ACM Transactions on Information Systems 7 (1) 3–29.

[16] M. A. S. Turine, M. C. F. de Oliveira, P. C. Masiero, A navigation-oriented hypertext model based on statecharts, in: HYPERTEXT '97: Proceedings of the eighth ACM conference on Hypertext, 1997, pp. 102–111.

[17] M. D. Zisman, Use of Production Systems for Modelling Asynchronous, Concurrent Processes, Pattern-Directed Inference Systems (1978) 53–68.

[18] W. R. van Biljon, Extending Petri nets for specifying man-machine dialogues, Int. J. Man-Mach. Stud. 28 (4) (1988) 437–455.

[19] P. D. Bra, G.-J. Houben, H. Wu, AHAM: a Dexter-based reference model for adaptive hypermedia, in: Proceedings of the tenth ACM Conference on Hypertext and hypermedia: returning to our diverse roots, 1999, pp. 147–156.

[20] S. Casteleyn, O. D. Troyer, S. Brockmans, Design time support for adaptive behavior in Web sites, in: Proceedings of the 2003 ACM symposium on Applied computing, 2003, pp. 1222–1228.

[21] N. Koch, M. Wirsing, The Munich Reference Model for Adaptive Hypermedia Applications, in: Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Springer-Verlag, 2002, pp. 213–222.

[22] P. Brusilovsky, Adaptive Hypermedia, User Modeling and User-Adapted Interaction 11 (1-2) (2001) 87–110.

[23] M. Cannataro, A. Pugliese, A Survey of Architectures for Adaptive Hypermedia, in: M. Levene, A. Poulovassilis (Eds.), Web Dynamics, Springer-Verlag, Berlin, 2004, pp. 357–386.

[24] S. Casteleyn, O. D. Troyer, S. Brockmans, Design time support for adaptive behavior in Web sites, in: Proceedings of the 2003 ACM symposium on Applied computing, 2003, pp. 1222–1228.

[25] A. Kobsa, Generic User Modeling Systems., User Model. User-Adapt. Interact. 11 (1-2) (2001) 49–63.

[26] F. Halasz, M. Schwartz, The Dexter Hypertext Reference Model, Comm. of the ACM 37 (2) (1994) 30–39.

[27] P. D. Bra, A. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, N. Stash, AHA! The adaptive hypermedia architecture, in: HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia, ACM Press, New York, NY, USA, 2003, pp. 81–84.

[28] O. D. Troyer, Audience-driven Web Design, Idea Group, 2001, pp. 442–462.