Combining Conceptual Modeling and Active Rules for the Design of Adaptive Web Applications

Florian Daniel   Maristella Matera   Giuseppe Pozzi

Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32 – 20133 Milano – Italy
{daniel,matera,pozzi}@elet.polimi.it

ABSTRACT

In this paper we propose a framework for the design and development of adaptive Web applications. The framework leverages on the integration of two well established approaches: a conceptual model, complemented with a CASE tool for automatic code generation, and a language for expressing ECA rules, supported by an engine for rule execution. Such integration leads to a versatile and flexible adaptivity environment, whose advantage is twofold: on one hand, conceptual modeling and automatic code generation support an efficient development process; on the other hand a detached rule engine allows us to widen the set of adaptivity requirements that can be handled and to overcome some limitations of current modeling approaches.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering*; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/ Hypermedia; H.1 [**Information Systems**]: Models and Principles

General Terms

Design, Languages.

Keywords

Adaptivity, Context-Awareness, Adaptive Web Applications, Web Application Modeling, Active Rules.

1. INTRODUCTION

More and more Web application users ask for services and contents highly tailored to their particular contexts of use. Especially due to the increasing affordability of new and powerful mobile communication devices, they also appreciate the availability of ubiquitous access, independent from the device actually in use. Due to such premises, traditional software design methods will no longer be exhaustive, and new issues and requirements need to be addressed to support adaptive access to services and applications. In order to react to this challenge, well-established design methods [9, 10, 1, 13, 4] have been extended to cope with adaptivity and context awareness. However, often the extensions do not cover the broad range of events that can trigger adaptivity, and do not address all the Web application dimensions that can be affected by adaptivity. Furthermore, in most cases mechanisms for managing adaptivity are buried within the application code, thus negatively affecting separation of concerns, and hindering maintenance and evolution.

We propose a framework for the development of adaptive Web applications. The framework leverages on the integration of two well-established approaches: the *Web Modeling Language* (WebML) method, based on a conceptual model [5] and a CASE tool for automatic code generation [16], and the *Chimera-Exception* language for expressing Event-Condition-Action (ECA) rules, supported by an engine for rule execution [2]. WebML is used to design Web applications: its recent extension to adaptivity modeling [3, 4] is adopted to specify *localized* adaptivity rules, i.e., rules attached to pages, which are triggered by changes in a "page context" monitored while pages are accessed. *Chimera-Web*, a new version of Chimera-Exception addressing Web events and Web adaptivity actions, is then used to specify *sparse* adaptivity rules, whose execution is completely detached from the execution of the application itself and whose effects are not necessarily bound to instances of modeling constructs.

Such an integration leads to a framework that keeps the advantages of conceptual modeling and automatic code generation, since the two dimensions, application design and adaptivity design, are handled at a high level of abstraction. The availability of a detached rule engine widens the set of events to react to, also comprising events that are independent from user actions. The resulting architecture enhances separation of concerns, and supports flexibility and evolvability: thanks to the availability of a detached rule engine, the modification or the addition of rules can be managed even after application deployment through Chimera-Web, without requiring changes to the application design and the generation of a new application version.

The paper is organized as follows: Section 2 discusses the rationale behind our work, and depicts the basic concepts of WebML and Chimera-Exception. Section 3 introduces the extensions of the two languages supporting localized and sparse adaptivity; a case study also exemplifies the new concepts. Section 4 then sketches the architecture of the integrated framework, and also provides some details about its current implementation. Section 5 finally draws our conclusion and outlines our future work.

2. RATIONALE AND BACKGROUND

Adaptivity is the ability of a system to react to changes in the user profile, the user device, and any attribute of the usage environment demanding for modifications of the offered contents and services. Web applications can be exposed to a multitude of adaptive behaviors. Based on the scope of the rules handling such behaviors, it is possible to distinguish between localized and sparse adaptivity. *Localized* adaptivity is strictly coupled with some hypertext elements (e.g., pages, links, etc.), as it happens in the case of an automatic update of the contents published by a particular page as reaction to a change in the user profile or in the usage environment. *Sparse* adaptivity requirements, on the other hand, may be bound to several hypertext/application components or may have no specific binding at all. An adaptation of the overall application's presentation properties, for example in response to a change in luminosity, represents a sparse adaptivity action.

Several well-established design methods have been so far extended to deal with Web application adaptivity. In [9] the authors extend the Hera methodology with two kinds of adaptation: adaptability with respect to the user device and adaptivity based on user profile data. Adaptation rules (and the Hera schemas) are expressed in RDF(S), attached to slices and executed by the AHA engine [7]. The UWA Consortium [15] proposes WUML [13] for conceptual hypertext design. Adaptation requirements are expressed by means of OCL-based customization rules referring to UML class or package elements. In [1] the authors explore *Aspect-Oriented Programming* [8] techniques for modeling adaptivity in the context of the UML-based Web Engineering method UWE [14]. Finally, in [10] the authors enrich the OO-H method with personalization rules for profile groups: rules are defined in PRML (Personalization Rule Modeling Language) and attached to links in the OO-H Navigation Access Diagram. The use of a (PRML) rule engine is envisioned in [11], but its real potential for adaptivity remains unexplored.

This paper capitalizes on the experiences of the previous works and proposes a solution based on conceptual modeling and active rules, respectively addressing different adaptivity requirements. For modeling the application front-end, we adopt the Web Modeling Language (WebML) [5] along with its recent extension to specify localized adaptivity rules. We then revise Chimera-Exception, an active rule definition language for specifying expected exceptions in workflow management systems (WfMSs), for handling sparse adaptivity rules.

The rest of this section introduces the basic concepts of the WebML method and the Chimera-Exception language. The adaptivity extensions of the two languages are described in Section 3; their integration is then described in Section 4.

2.1 WebML

WebML [5] is a visual language for specifying the content structure of a Web application and the organization and presentation of contents in one or more hypertexts.

The design process starts with the specification of a data schema, expressing the organization of the contents of the Web application. The *WebML data model* uses Entity–Relationship primitives. The *WebML hypertext model* then allows one to describe how the content, previously specified in the data schema, is published in the application hypertext. The overall structure of the hypertext is defined in terms of *site views*, *areas*, *pages*, and *content units*. A *site view* is a hypertext, designed to address a specific set of requirements. Several site views can be defined on top of the same data schema, to serve the needs of different user communities, or to arrange the composition of pages to meet the requirements of different access devices such as PDAs, smart phones, and similar appliances. A site view is composed of *areas*, which are the main sections of the hypertext and comprise, recursively, other subareas or pages. *Pages* are the actual containers of information delivered to the user; they are made of *content units*, which are the elementary pieces of information extracted from the data sources by means of queries, and published within pages.

Content units and pages are interconnected by *links* to build site views. Links can connect units in a variety of configurations, yielding to composite navigation patterns. Besides representing user navigation, links between units also specify the transportation of some information used by the destination unit to select the data instances to be displayed.

Some WebML units also support the specification of content management operations. They allow one to create, delete or modify an instance of an entity (through the `create`, `delete` and `modify` units respectively), or to add or drop a relationship between two instances (through the `connect` and `disconnect` units, respectively). Recently, WebML has also been extended to model invocations of Web services; in this context, application data can be derived from external data sources as well. For a more complete presentation of WebML and its visual notation, the reader is referred to [5].

Besides having a visual representation, WebML primitives are provided with an XML-based textual representation, which specifies additional detailed properties, not conveniently expressible in the graphic notation. Web application design based on WebML can be therefore represented as visual diagrams, as well as XML documents. The XML representation constitutes the starting point for the automatic generation of the application code to be executed by means of a proper runtime environment [16].

2.2 The Chimera-Exception Language

Chimera-Exception [2] is derived from *Chimera* [6], a language to describe active rules in database management systems. Chimera-Exception builds on an object-oriented process definition formalism, where classes are typed and represent records of typed attributes that can be accessed by means of a simple dot-notation. Classes are either *workflow-independent* (e.g. role, agent, task) if they are independent from the structure of the process, or *workflow-dependent* (e.g. workflow-relevant data fields) if instead they depend on a specific process definition.

The Event-Condition-Action (ECA) constructs specify active rules, adhering to the following trigger structure:

```
define trigger <TriggerName>
      [for <ProcessSchema> | global]
  events <Event> [(, <Event>)+]
  condition [<Cond> [(, <Cond>)+] | none]
  actions <Action> [(, <Action>+)]
  [order <PriorityValue>]
end
```

A trigger `<TriggerName>` has one or more triggering events (`<Event>`), a condition possibly comprising several (or `none`) conditional statements (`<Cond>`), and one or more actions (`<Action>`) describing the workflow-specific actions to be performed in case the condition of the triggered rule holds. Rules have a statically defined priority (`<PriorityValue>`), specified as a positive integer, which is used to determine the order of execution within a set of triggered rules: the higher the number, the higher the priority.

Rules can be triggered by several types of *events*. *Data events* refer to changes of workflow-relevant data fields or of the underlying data source containing the process information model. *Workflow events* are related to the start and the completion of tasks and cases. *External events* are generated by (external) applications and require suitable handling mechanisms *Temporal events* can be synchronous or asynchronous with respect to the process execution. *Conditions* consist of predicates that inspect the content of the database or of workflow-relevant data fields. *Actions* comprise assigning a task or a case to an agent (`assignTask(TaskId)`, `assignCase(CaseId)`), canceling a task or a case (`cancelTask(TaskId)`, `cancelCase(CaseId)`), start a task or a new case, suspend a task, etc.

If a rule is defined for a particular `<ProcessSchema>`, the rule is triggered by events related to instances of that specific schema, only. If instead the rule is defined as `global`, its scope extends to the whole execution environment, and thus comprises all the available process definitions. A more detailed description of the language is available in [2, 12].

Chimera-Exception is also complemented with an active rule engine (FAR) integrated with a commercial WfMS (FORO) and an active database server [2, 12]. The FAR system consists of a rule compiler, a time manager, a scheduler and a runtime interpreter, which together enable the execution of Chimera-Exception rules.

3. A TWO-LEVEL APPROACH TO ADAPTIVITY SPECIFICATION

We now show how WebML and Chimera-Exception can be adapted and combined to support the specification of a two-level adaptivity, covering *localized* and *sparse* adaptivity requirements. In particular:

- The *WebML* extension to adaptivity [3, 4] is used for coping with localized adaptivity, which follows the WebML design and computation logic, centered around the page concept. WebML has been extended (i) at data level for modeling user and context meta-data, and (ii) at hypertext level for specifying localized adaptivity actions associated to hypertext pages.
- *Chimera-Exception* is revised to support the specification of sparse adaptivity. In particular, workflow-related primitives are substituted by Web-specific primitives, especially needed for capturing Web events and executing the corresponding Web adaptivity actions. We call the resulting extension *Chimera-Web*.

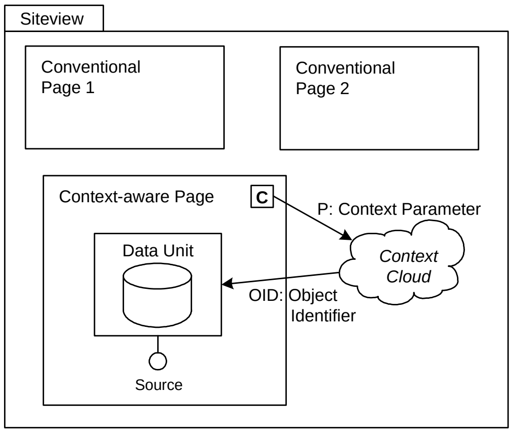

Figure 1: Specification of adaptivity actions in WebML.

3.1 WebML and Localized Adaptivity

In [3, 4], WebML is extended to cover localized adaptivity requirements. The overall design process for adaptive Web applications follows the activity flow typically used for conventional Web applications. However, capturing adaptivity implies modeling the user and the context of interaction. Therefore, during data design, the user and context requirements can be translated into three different subschemas complementing the application data:

- The *user sub-schema*, which clusters data about users and their access rights to application data.
- The *personalization subschema*, which consists of entities from the application data, associated with the `User` by means of relationships expressing user preferences or rights for some entity instances. In general, relationships defined between the entity `User` and any other entity of the application data support the personalization of the content of the entity with respect to the identity of the user.
- The *context sub-schema*, which includes entities that describe particular properties of the context (such as `Device`, `Location` and `Activity`). Context entities are connected to the entity `User` to associate each user with her/his (personal) context.

The previous representation of user and of context meta-data enables the specification at hypertext level of *localized adaptivity rules* that are attached to pages. Rules are evaluated on a *page context*, i.e., a page-specific set of attributes within the data source, which are monitored for triggering a page's rule while the page is visited. Pages provided with rules are tagged with a `C`-label standing for "Context-aware", due to the association of the page with its *page context*. As illustrated in Figure 1, the specification of adaptivity rules is kept outside the page, within the so-called *context-cloud*, aimed at highlighting the two different roles played by pages and adaptivity actions: while pages act as providers of content and services, the adaptivity actions act as modifiers of such content and services. As better discussed in Section 4, while accessing a `C`-page, the state of the page context is observed to detect any change demanding for adaptivity. When occurred, such changes generate the automatic request of the page and the evaluation of the page's context cloud.

In WebML rules, *events* are not explicitly modeled, as they correspond to modifications of the variables declared in the page context. Some WebML constructs (i.e., operation units and links) are instead needed to visually define conditions and actions. In particular, *conditions* are modeled by conditional constructs, such as `If` and `Switch` units. If the conditions are satisfied, several *actions* can be performed:

- *Page content adaptivity.* Context parameters, acquired through the user device or through a dedicated sensing infrastructure, as well as parameters computed during condition evaluation, can be used for page computation. The result is a page where contents are *filtered* with respect to the current context.
- *Navigation adaptivity.* In some cases, the effect of condition evaluation within the context cloud can be an automatic, *context-triggered* navigation, causing the redirection to a different page. Links exiting the context cloud and directed to pages other than the cloud's source page represent *automatic navigation actions*. Also, some values generated by the evaluation of conditions may imply updating the current page by hiding or showing navigation links.
- *Hypertext structure adaptivity.* This allows one to face coarse-grained adaptivity requirements, as required, for example, in the case of changes of the user's device, role and/or activity within a multi-channel, mobile environment. A new `Change Site View` construct allows switching between site views [3, 4].
- *Presentation adaptivity.* More fine-grained adjustments of the application's appearance can be achieved by the `Change Style` construct [3, 4], which allows changing at runtime the page's CSS (*Cascading Style Sheet*) file that codes the page's presentation properties.
- *Back-end operations.* Besides the previous actions, mainly referred to modifications of the application front-end, WebML rule actions may also specify back-end operations and invoke external (Web) services.

3.2 Chimera-Web and Sparse Adaptivity

Chimera-Web is the extension of Chimera-Exception to manage Web events and Web actions supporting Web application adaptivity. Analogously to Chimera-Exception, rules in Chimera-Web are triggered by four main kinds of *events*:

- *Data events* refer to operations on the application data source, such as `create`, `modify` and `delete`. In adaptive Web applications, such events can be monitored on user, customization and context data, to trigger adaptivity actions with respect to users and their context of use.
- *Web events* replace the *Workflow events* of Chimera-Exception. Web events refer to general browsing activities (e.g., `pageAccess(Page)`, `formSubmit(Form)`, `pageRefresh(Page)`, `download(Href)`), or to operations supported by the application (e.g., `operationStart(Op)`, `OperationEnd(Op)`, `login`, `logout`).
- *External events* are recognized by the `raise` primitive, providing, as an external event occurs, the name of the triggering event and, possibly, suitable parameters. External events can be generated both locally or remotely with respect to the Web application server.
- *Temporal events* are subdivided into *instant*, *periodic* and *interval events*. Interval events are particularly powerful, since they allow binding a time interval to other events. For example, the expression `elapsed(interval 5 minutes) since pageAccess(Page1)` represents a temporal event that is raised after the expiration of 5 minutes from the access to page `Page1`.

In Chimera-Web, *conditions* consist of predicates over context data, application data, global session variables and page parameters. As in WebML, *actions* in Chimera-Web correspond to modifications of the application front-end, as well as to the execution of back-end operations. Suitable predicates are introduced for specifying such actions.

3.3 WebML versus Chimera-Web

WebML rules can be expressed through an intuitive visual language. Their specification is performed in the same environment where the application is designed, with the same paradigm adopted for page design. The new visual modeling constructs have been implemented as extensions to the WebML CASE tool and the WebML runtime environment [16]. This allows covering the whole (visual) development process, from design to automatic code generation and to the execution of localized adaptivity rules.

Despite the previous advantages, WebML rule specification has however some drawbacks, which justify the introduction of Chimera-Web in our integrated framework. In particular:

- WebML adaptivity actions are coupled with pages.
- At a given time, the set of rules the application is able to react to is limited to the rules attached to the page the user is currently visiting.
- Given a page, the specification of a complex chain of rules, possibly acting on different application dimensions (e.g., data, navigation, and presentation), could

not be trivial and might lead to cognitive overloaded diagrams that negatively impact on the scalability of the modeling paradigm.

- In order to be taken into account by several pages, a WebML adaptivity rule must be attached to every page it applies to. Such redundancy reduces schema readability and does not support reuse.
- Furthermore, the *page context*, as originally conceived, applies only to data stored in the application data source, thus not considering temporal or external events.

Similar limitations can be found (with different facets) in most model-driven approaches. We therefore propose Chimera-Web for addressing the previous lacks, also introducing the following advantages:

- Rules may have a sparse scope, not necessarily limited to single hypertext pages. This facilitates the definition of rules spanning several pages and, more in general, several application components. Reuse and consistency are thus enhanced.
- The execution of Chimera-Web rules is detached from the computation of pages. The two logics, page computation and rule evaluation, are kept separate, thus enhancing separation of concerns even during application execution.
- Chimera-Web scales better with respect to the complexity of the rule chains to be attached to individual pages.
- Chimera-Web offers a broader set of events.
- The detached rule engine also facilitates the dynamic management of adaptivity rules: rules can be modified, newly introduced or dropped even during application execution. On the opposite, changes to WebML rules require the generation and deployment of a new version of the application code. Chimera-Web thus supports the evolution of adaptivity requirements.

3.4 Example

In the context of the Italian research project MAIS[1] we have developed a context-aware Web application, called *PoliTour*, supplying information about buildings and roads within our university campus at Politecnico di Milano. The application is accessed by a PDA, equipped with a GPS receiver for location sensing. User positioning is thus based on geographical longitude and latitude. As the user moves around the campus, the application publishes location-aware data, providing details about roads and buildings.

Figure 2 depicts the application data schema, where the entity `User` represents the user model and the entity `Position`, associated to `User`, constitutes the context model. The campus area can be divided into a set of contiguous, rectangular zones (modeled by means of the entity `Location`), and roads and buildings can be mapped onto different contiguous areas. Therefore, four attributes characterize the boundaries

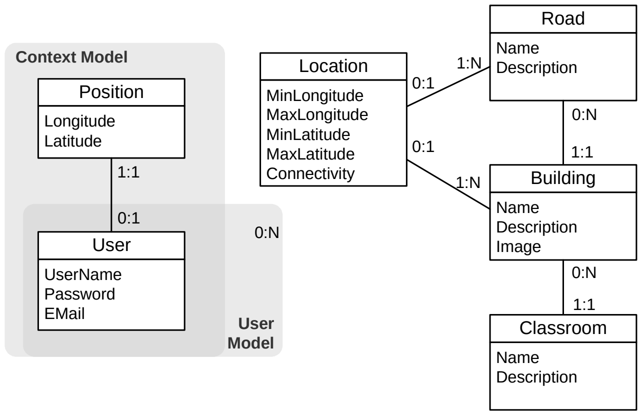

Figure 2: Application data of the *PoliTour* Web application.

of a location: min and max longitude, min and max latitude. Also, each location is associated with a measure of the expected quality of the wireless connection active in each location (attribute `Connectivity`).

Figure 3 shows a simplified WebML hypertext schema, where we only show three pages of the overall application. Page `Buildings` publishes a list of buildings (`BuildingsIndex`). It also shows details of a selected building (`BuildingData`) chosen from the list, together with the building's classrooms (`ClassroomsIndex`). The selection of a classroom leads the user to a new page (`Classroom`), which shows some details about the selected classroom (`ClassroomData`). Similarly to page `Buildings`, page `Roads` shows data about the campus roads (`RoadsIndex` and `RoadData`) and their nearby buildings (`Nearby Buildings`).

The pages `Buildings` and `Roads` are `C`-pages; the external unit chain represents their adaptivity rule. In particular, the unit `Get User` provides the identifier of the current user in input to the `Get Position` unit, which accesses location data sensed through the client-side GPS module. The `Get Location` and `Get Building` units retrieve the location corresponding to the sensed position and the contained building. If no building is retrieved, the `Get Road` unit retrieves the road associated with the current position, and forwards the user to the `Roads` page. Alternatively, if a building is retrieved, page `Buildings` is displayed. It is worth noticing that the two `C`-pages share the same adaptivity actions.

The application has been modeled according to the adaptive WebML approach, automatically generated with the extended WebML code generator and deployed on top of a J2EE platform, extended for supporting adaptivity. Its intended use occurs through PDA devices with wireless Internet access, using *Pocket Internet Explorer*[2]. The communication with the GPS module is implemented using the *Chaeron GPS Library*[3].

The visually modeled adaptivity logic in the described context-aware Web application allows achieving *localized* adap-

[1] `http://www.mais-project.it`

[2] A demo of the PoliTour application is available at `http://dblambs.elet.polimi.it/politour/`.

[3] `http://www.chaeron.com/gps.html`

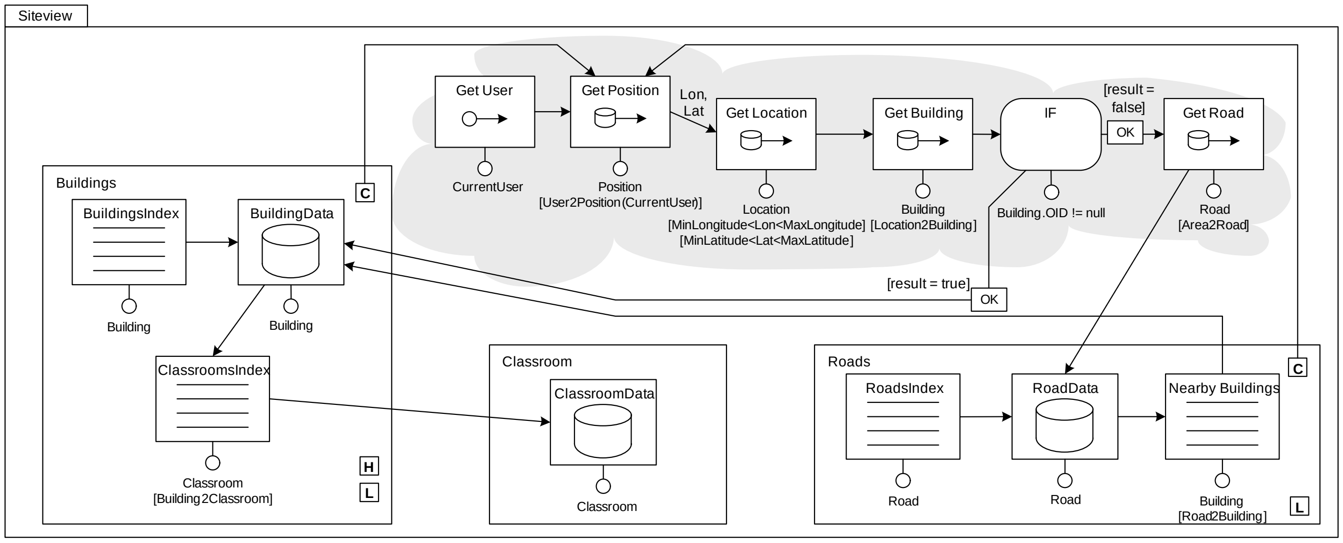

Figure 3: WebML hypertext model of the *PoliTour* application.

tivity effects. Indeed, the adaptivity actions shared by the two pages `Buildings` and `Roads` describe an adaptive behavior (i.e., reaction to changes in position data), which is proper of just these two pages.

3.4.1 *Specifying Sparse Adaptivity Rules*

As an example of sparse or cross-application adaptivity expressed by Chimera-Web, we consider the following `LowConnectivity` trigger. It is defined as `global`, and therefore refers to all the pages of the Web application.

```
define trigger LowConnectivity global
events    modify(Position.Latitude),
          modify(Position.Longitude)
condition Location(L),
          L.MinLatitude < Position.Latitude,
          Position.Latitude < L.MaxLatitude,
          L.MinLongitude < Position.Longitude,
          Position.Longitude < L.MaxLongitude,
          L.Connectivity = "Low",
          CurrentStyle != "LowConnStyle.css"
actions   changeStyle("LowConnStyle.css")
          order 1
end
```

The trigger is evaluated every time a change in the user position occurs, as indicated by the two (disjunctive) `modify` events. The evaluation of the trigger is independent from the pages currently viewed by users. According to the condition statement, the trigger is executed only if the new reached position falls inside a location with a “low” connectivity. The location corresponding to the current position is selected by means of the condition predicates and bound to the variable `L`, which joins the single predicates of the condition statement. As the action of the rule consists in changing the Cascading Style Sheet (CSS) associated with the page, a proper session parameter `CurrentStyle` indicates whether the action has already been performed or not. The aim of the new style sheet is that of alerting the user of possible connectivity problems, by changing the application's color composition.

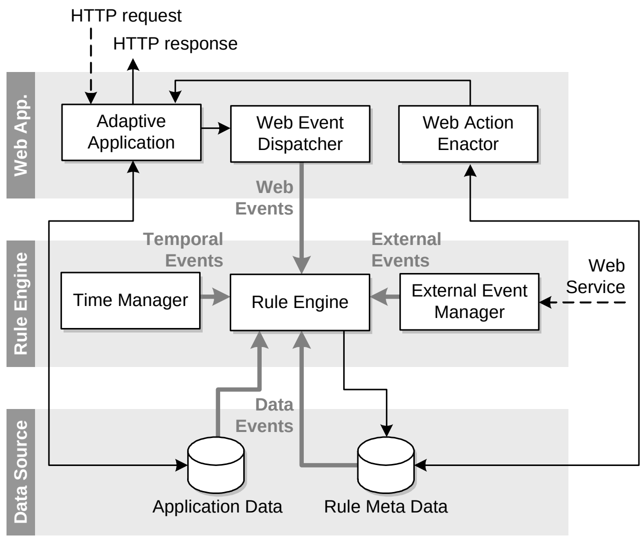

Figure 4: Layered system architecture articulated into *Web Application* layer, *Rule Engine* layer and *Data Source* layer.

4. THE INTEGRATED FRAMEWORK ARCHITECTURE

Figure 4 depicts a three-layer architecture for the integration of WebML and Chimera-Web. With respect to conventional Web applications and to [3, 4], this architecture also includes an intermediate *Rule Engine* layer, which is devoted to the detached execution of adaptivity rules. The three different layers aim at the following purposes:

- The *Data Source* layer contains both the application data and the meta-data required for rule evaluation and execution. With respect to conventional Web applications, the use of ECA rules requires an active database system, supporting the definition of triggers.

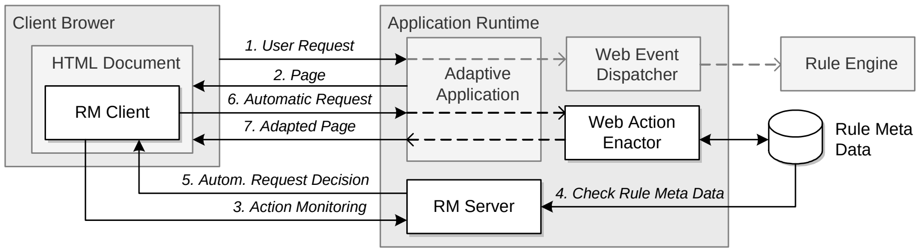

Figure 5: Enacting Web actions with client-side adaptivity effects.

- The *Rule Engine* layer supports event handling and rule execution. The input to the *Rule Engine* consists of temporal and external events (managed by the *Time Manager* module and the *External Event Manager* module, respectively), of Web events forwarded by the *Web Event Dispatcher*, and of data events corresponding to modifications to the data source.
  The execution of Chimera-Web rules within the *Rule Engine* layer requires their *a-priori* compilation that generates suitable meta-data, native trigger definitions in the adopted DBMS, and configures the *Time Manager* and the *External Event Manager*.
  Thanks to the modular architecture of the integrated framework, the rule engine can be switched on or off, depending on whether the execution of sparse rules is required or not.

- The *Web Application* layer is represented by the extended WebML runtime environment, which manages the application logic (*Adaptive Application* module). The integration with the Chimera-Web engine further requires the inclusion within the *Web Application* layer of the *Web Event Dispatcher* module, to capture Web events generated during the execution of the adaptive application, and of the *Web Action Enactor* module, to execute adaptivity actions. Both modules can be implemented as plug-ins for the extended WebML runtime environment.

As shown in Figure 4, events possibly triggering adaptivity are propagated directly (by their handling modules) or indirectly (via the *Rule Engine*) toward the rule meta-data. Such meta-data contain the "runtime" data of Chimera-Web rules, and are observed by the *Rule Engine* to evaluate triggered rules. If the condition of a triggered rule is satisfied, the engine activates the specified action handler(s) in the *Web Application* layer to enact the respective adaptivity action(s). Activating an action handler may imply a direct communication with the respective software module (e.g. by calling a function or method of the action handler) or the appropriate modification of rule meta-data, in case the required action handler modules themselves are able to observe the data source.

With respect to traditional trigger applications, Chimera-Web rules enable the execution of Web actions, that in some cases may also affect the Web application front-end. The execution of such actions requires some additional mechanisms to dynamically re-compute the page inspected by the user. Figure 5 depicts our solution to manage the execution of adaptivity actions with client-side effects.

The adoption of standard Web technologies prohibits the use of push mechanisms to communicate adaptations to be executed at the client side. Nevertheless, a client-server component added to adaptive pages, the so-called *Rule Monitor* (RM) allows us to simulate the required active behavior by means of a background polling solution, leading us to promote adaptivity rules as "active actors", operating independently from users on the same hypertext the users navigate. The *RM Server* monitors the rule meta-data, looking for modifications. Periodically, the *RM Client* (an active client-side component added to the HTML markup of adaptive pages[4]) consults the *RM Server* to identify whether any modification occurred. In case of modification, the *RM Client* generates an automatic page refresh, allowing the *Web Action Enactor* to apply the required adaptations when computing the page.

4.1 Managing Rule Conflicts

In order to avoid possible conflicts among rules triggered at the two adaptivity levels, a unified logic for rule triggering is required. Such a unified logic is achieved by referring the execution of visually defined rules to the Chimera-Web rule engine. Since the rule engine is based on the monitoring of rule meta-data, a unified management of rules requires the introduction of meta-data representing any change on the page context and the definition of triggers for identifying such changes. More precisely, each visual definition of a rule requires the specification of the data attributes in the page context. Starting from page context data, the automatic generation of the application code produces a Chimera-Web trigger that is capable of reacting to data events over the page context. In case any modification to such attributes occurs, the trigger sets some rule meta-data to notify the RM that the rule's actions are to be performed.

Also, a sensible use of priorities associated to rules (see Section 2.2) allows ordering both visual and textual rules, according to their priority. Analogously to page context attributes, also the priority of visual rules can be set through a page parameter within the WebML CASE tool. Conflict resolution and priority management are thus completely delegated to the *Rule Engine* layer and do not impact on the runtime environment of the Web application. As for the Chimera-Exception language, a suitable TAM (Termination Analysis Machine) is invoked at rule definition time to check for possible non-termination [2].

[4]In our current experiments, the RM client is implemented as Flash object, while the RM Server is a Java servlet.

4.2 Unbundling the ECA Server

For the execution of Chimera-Web rules, we are currently unbundling and generalizing the active component of Chimera-Exception from its original WfMS. The so obtained ECA server will then be fully integrated with the extended WebML runtime environment as described in Figure 4.

In fact, we aim at unbundling the exception handling unit FAR from FORO. FAR detects data events by means of rules at the database level, while workflow events are related to the engine of the WfMS itself and, possibly, are detectable at the database level. Temporal events, which are set up by an anchor event or by the absolute (GMT) time, are managed by a suitable module (TimeManager) interfaced with the system clock. External events are triggered by external programs, which must register inside FAR and can then signal the occurrence of the respective event. Thus, the coupling between FAR and the WfMS is tight for data and workflow events, while it is more relaxed for temporal and external events. Unbundling the ECA server therefore implies eliminating domain-specific events in FAR (i.e., workflow events), while the integration with the WebML runtime environment (i.e., the re-bundling of the ECA server), as described in Figure 4 expects the definition of new domain events (i.e., Web events). In order to avoid modifying the compiler unit to manage *Web adaptivity actions*, the solution proposed in this paper is the introduction of the *Web Event Enactor* module as an external executor.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have considered a relevant aspect of modern Web applications, i.e. adaptivity. We have shown how such issue can be addressed by a two-layered approach that combines conceptual modeling with active rules and yields an adequate framework for coping with both localized and sparse adaptivity requirements.

With respect to our previous results, the integrated framework described in this paper provides a full-fledged support for any type of events, internal or external to the application, dependent or independent from user interactions, and broadens the scope of adaptivity rules. Although Chimera-Web has been introduced for managing sparse adaptivity rules, it can be used as well to specify and execute localized rules. This aspect further enhances the evolvability of localized rules. Indeed, localized rules can be "dynamically" extended after the application deployment, without requiring modifications to the conceptual specification of the application, which would require a new code generation and a new deployment.

The simulations we already completed on a "loosely coupled" architecture proved the feasibility of the outlined integration. We are now working on implementing the additional modules the integration is based on, namely the Web Action Enactor and the Web Event Dispatcher.

6. REFERENCES

[1] H. Baumeister, A. Knapp, N. Koch, and G. Zang. Modeling Adaptivity with Aspects. In D. Lowe and M. Gaedke, editors, *Proc. of ICWE 2005, Sydney, Australia.*, volume 3579 of *LNCS*, pages 406–416. Springer-Verlag Berlin Heidelberg, July 2005.

[2] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Transactions on Database Systems*, 24(3):405–451, 1999.

[3] S. Ceri, F. Daniel, and M. Matera. Extending WebML for Modeling Multi-channel Context-aware Web Applications. In *Proc. of WISE'03 Workshops, Rome, Italy, December 12 -13, 2003*, pages 225–233. IEEE Press, 2003.

[4] S. Ceri, F. Daniel, M. Matera, and F. M. Facca. Model-driven Development of Context-Aware Web Applications. *ACM Transactions on Internet Technologies*, 7(2), 2007.

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

[6] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Active Rule Management in Chimera. In *Active Database Systems: Triggers and Rules For Advanced Database Processing*, pages 151–176. Morgan Kaufmann, 1996.

[7] P. De Bra, A. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash. AHA! The Adaptive Hypermedia Architecture. In *Proc. of HYPERTEXT '03*, ACM Press, pages 81–84, 2003.

[8] R. E. Filman, T. Elrad, S. Clarke, and M. Aksit. *Aspect-Oriented Software Development*. Addison-Wesley, 2004.

[9] F. Frasincar and G.-J. Houben. Hypermedia Presentation Adaptation on the Semantic Web. In *AH*, pages 133–142, 2002.

[10] I. Garrigós, S. Casteleyn, and J. Gómez. A Structured Approach to Personalize Websites Using the OO-H Personalization Framework. In *Web Technologies Research and Development - APWeb 2005*, pages 695–706. Springer-Verlag, 2005.

[11] I. Garrigós, J. Gómez, P. Barna, and G.-J. Houben. A Reusable Personalization Model in Web Application Design. In *WISM'05*, 2005.

[12] P. Grefen, B. Pernici, and G. Sanchez, editors. *Database Support for Workflow Management: The Wide Project*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[13] G. Kappel, B. Pröll, W. Retschitzegger, and W. Schwinger. Modelling Ubiquitous Web Applications - The WUML Approach. In *ER (Workshops)*, pages 183–197, 2001.

[14] N. Koch, A. Kraus, and R. Hennicker. The Authoring Process of the UML-based Web Engineering Approach. In D. Schwabe, editor, *Proc. of IWWOST'01*, 2001.

[15] UWA Consortium. The UWA Approach to Modeling Ubiquitous Web Applications. *IST Mobile and Wireless Telecommunications Summit*, 2002.

[16] WebModels s.r.l. Webratio Site Development Studio. http://www.webratio.com, 2005.