Active Rules for Runtime Adaptivity Management

Florian Daniel, Maristella Matera, Alessandro Morandi, Matteo Mortari, and Giuseppe Pozzi

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
{daniel,matera,morandi,mortari,pozzi}@elet.polimi.it

Abstract. The trend over the last years clearly shows that modern Web development is evolving from traditional, HTML-based Web sites to full-fledged, complex Web applications, also equipped with active and/or adaptive application features. While this evolution unavoidably implies higher development costs and times, such implications are contrasted by the dynamics of the modern Web, which demands for even faster application development and evolution cycles.
In this paper we address the above problem by focusing on the case of adaptive Web applications. We illustrate an ECA rule-based approach, intended to facilitate the management and evolution of adaptive application features. For this purpose, we stress the importance of decoupling the active logic (i.e. the adaptivity rules) from the execution of the actual application by means of a decoupled rule engine that is able to capture events and to autonomously enact adaptivity actions.

1 Introduction

Adaptability (the design-time adaptation of an application to user preferences and/or device characteristics [1]) and *adaptivity* (the runtime adaptation of an application to a user profile or a context model [1]) have been studied in the last years by several authors in the field of Web engineering. Adaptability is intended as the capability of the design to fit an application to particular needs prior to the execution of the application. Adaptivity is intended as autonomous capability of the application to react and change in response to specific events occurring during the execution of the application, so as to better suit dynamically changing execution conditions. Recently, adaptivity has been extended to the case of *context-aware* Web applications [2], where adaptivity is based on a dynamically updated context model, upon which the adaptive application is built.

As is the nature of the Web engineering discipline, the previous approaches to adaptability, context-awareness and adaptivity primarily focus on the definition of *design processes* to achieve adaptation, thereby providing efficient methods and tools for the design of such a class of applications. For instance, model-driven methods [1, 2], object-oriented approaches [3], aspect-oriented approaches [4],

and rule-based paradigms [5, 6] have been proposed for the specification of adaptation features in the development of adaptive Web applications. The resulting specifications facilitate the implementation of the adaptation requirements and may also enhance code coherence and readability. Unfortunately, in most cases during the implementation phase all the formalizations of adaptivity requirements are *lost*, and the adaptivity features become buried in the application code. This aspect implies that changes and evolutions of adaptive behaviors after the deployment of the application are difficult, unless a new version of the application is implemented and released.

Based on our experience in the model-driven design of adaptive/context-aware Web applications [2, 7], we are convinced that the next step in this research area is to support the *dynamic* management of adaptivity features: on one hand this will require proper design time support (e.g. languages or models), on the other hand this will require suitable runtime environments where adaptivity specifications can be easily administered.

In [8] we already outlined a first conceptual framework for this approach. We now focus on the evolution of that work, describing a rule-based language (ECA-Web) for the specification of adaptive behaviors, orthogonally to the application design, and its concrete implementation. The resulting framework provides application designers with the ECA-Web language and application administrators with the possibility to easily manage ECA-Web rules (inserting, dropping, and modifying rules), even after the implementation and the deployment of the application, i.e. at runtime. As envisioned above, by the described approach we enable the decoupled management of adaptivity features at both design- and run-time.

This paper is organized as follows. Section 2 discusses some related works on adaptivity in the Web. Section 3 introduces the ECA-Web rule language for the specification of adaptive behaviors for Web applications and, then, shows how ECA-Web rules can be executed by a proper rule engine and integrated with the execution environment of the adaptive Web application. Section 4 discusses the prototype of an adaptive Web application supported by ECA-Web rules and shows the usage of the active rule language. Section 5 describes the implementation of the overall system and reports on first experiences with the rule-based adaptivity specification and the runtime management of adaptivity rules. Finally, Section 6 concludes the paper and discusses future work.

2 Related Work

Conceptual modeling methods provide systematic approaches to design and deploy Web applications. Several well-established design methods have been so far extended to deal with Web application adaptivity. In [1] 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) (Resource Description Framework/RDF Schema), attached to slices and executed by the AHA engine [9].

The UWA Consortium proposes WUML [10] for conceptual hypertext design. Adaptation requirements are expressed by means of OCL-based customization rules, referring to UML class or package elements. In [11] the authors present an extension of WSDM [12] to cover the specification of adaptive behaviors. In particular, an event-based Adaptive Specification Language (ASL) is defined, which allows designers to express adaptations on the structure and the navigation of the Web site. Such adaptations consist in transformations of the navigation model, which can be applied to nodes (deleting/adding nodes), information chunks (connecting/disconnecting chunks to/form a node), and links (adding/deleting links). In [4] the authors explore Aspect-Oriented Programming techniques to model adaptivity in the context of the UML-based Web engineering method UWE. Recently, WebML [13] has been extended to cover adaptivity and context-awareness [2]. New visual primitives cover the specification of adaptivity rules to evaluate conditions and to trigger some actions for adapting page contents, navigation, hypertext structure, and presentation. Also, the data model has been enriched to represent some meta data supporting adaptivity.

The previous works benefit from the adoption of conceptual models, which provide designers with powerful means to reason at a high-level of abstraction, independently of implementation details. However, the resulting specifications of adaptivity rules have the limit of being embedded inside the design models, thus raising problems in the maintenance and evolution of the adaptivity requirements, once the application is released.

Recently, active rules, based on the ECA (Event-Condition-Action) paradigm, have been proposed as a way to solve the previous problem. Initially exploited especially in fields such as content evolution and reactive Web [14–16], ECA rules have been recently adopted to support adaptivity issues in Web applications. In particular, the specification of decoupled adaptivity rules provides a way to design adaptive behaviors along an orthogonal dimension. Among the most recent and notable proposals, the work described in [5] enriches the OO-H model with personalization rules for profile groups: rules are defined in PRML (Personalization Rule Modeling Language) and are attached to links in the OO-H Navigation Access Diagram. The use of a PRML rule engine is envisioned in [6], but its real potential for adaptivity management also at runtime remains unexplored.

In line with the previous work, the approach we describe here proposes a rule-based language adopting the ECA paradigm. We call the language ECA-Web, emphasizing that it is able to express events and actions that may occur in a Web environment. Although the proposed language allows one to reference elements of a conceptual specification of an application[1], it is a self-sufficient language for the specification of adaptivity rules. The novelty of our work is however the development of a decoupled environment for the execution and administration of adaptivity rules, which allows the management of adaptivity features to be kept totally independent of the application execution. This aspect introduces several advantages in terms of maintainability and evolvability.

[1] In this paper we shall briefly show how the language can be bound to WebML [13].

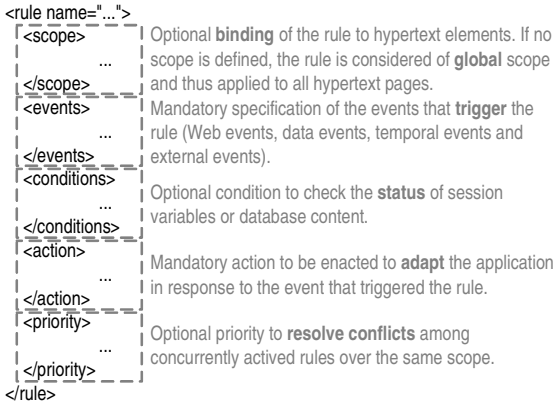

Fig. 1. Structure of ECA-Web rules.

3 Enabling Dynamic Adaptivity Management

In the following we introduce the *design* component (the ECA-Web language) and the *runtime* component (the rule execution environment) that enable the dynamic administration of adaptivity features.

3.1 ECA-Web

ECA-Web is an XML-based language for the specification of active rules, conceived to manage adaptivity in Web applications. The syntax of the language is inspired by Chimera-Exception, an active rule language for the specification of expected exceptions in workflow management systems [17]. ECA-Web is an evolution of the Chimera-Web language we already proposed in [8], and it is equipped with a proper rule engine for rule evaluation and execution.

The general structure of an ECA-Web rule is summarized in Figure 1. A typical ECA-Web rule is composed of five parts: scope, events, conditions, action and priority. The *scope* defines the binding of the rule with individual hypertext elements (e.g. pages, links, contents inside pages). By means of *events* we specify how the rule is triggered in response to user navigations or changes in the underlying context model. In the *condition* part it is possible to evaluate the state of application data (e.g. database contents or session variables) to decide whether the action is to be executed or not. The *action* specifies the adaptation of the application in response to a triggered event and a true condition. The *priority* defines an execution order for rules concurrently activated over the same scope; if not specified, a default priority value is assigned. More details on the rule specification by means of ECA-Web are given in the next section, where we discuss the architecture of the runtime environment for rule execution. An example of ECA-Web rule will then be shown in Section 4.

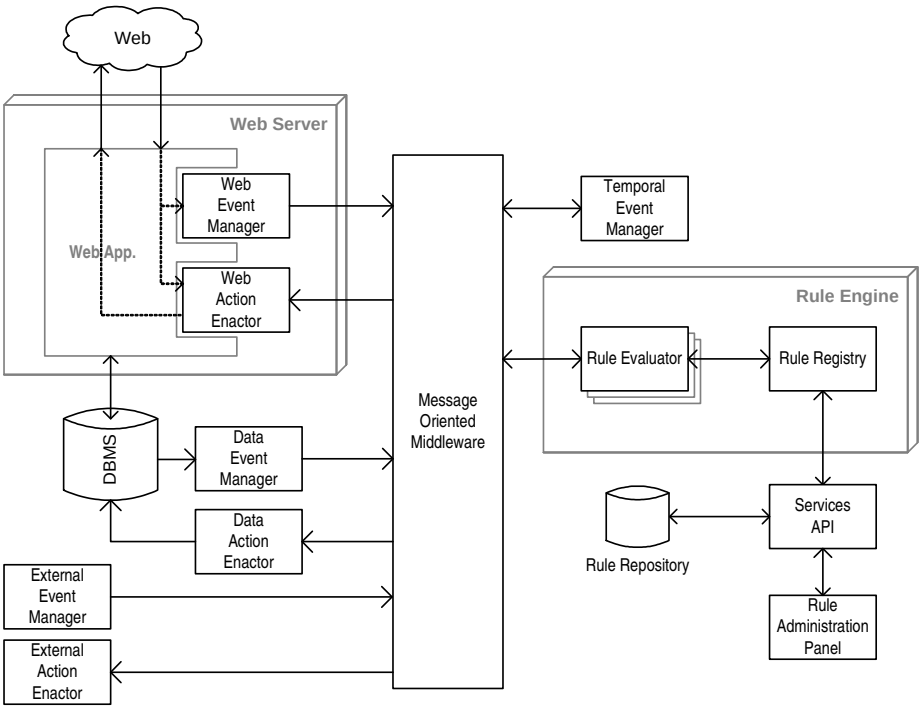

Fig. 2. Functional architecture of the integrated execution environment for adaptive Web applications.

3.2 The Integrated Runtime Architecture

The execution of ECA-Web rules demands for a proper runtime support. Figure 2 summarizes the functional architecture of the system, highlighting the two main actors: the *Rule Engine* and the *Web Server* hosting the Web application. The *Rule Engine* is equipped with a set of *Event Managers* to capture events, and a set of *Action Enactors* to enable the execution of actions. The communications among the single modules are achieved through asynchronous message exchanges (*Message-Oriented Middleware*).

Event Managers. Each type of ECA-Web event is supported by a suitable event manager (i.e., *Web Event Manager*, *Data Event Manager*, *Temporal Event Manager*, and *External Event Manager*). As in [8], event managers and ECA-Web provide support for the following event types:

– *Data events* refer to operations on the application's 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. Data events are

managed by the *Data Event Manager*, which runs on top of the application's data source.
- *Web events* refer to general browsing activities (e.g. the access to a page, the submission of a form, the refresh of a page, the download of a resource), or to events generated by the Web application itself (e.g. the start or end of an operation, a login or logout of the user). Web events are risen in collaboration with the Web application and captured by the *Web Event Manager*. Since adaptivity actions are typically performed for each user individually, Web events are also provided with a suitable user identifier (if any).
- *External events* can be configured by a dedicated plug-in mechanism in form of a Web service that can be called by whatever application or resource from the Web. An external event could be for example a notification of news fed into the application via RSS. When an external event occurs, the name of the triggering event and suitable parameters are forwarded to the rule engine. External events are captured by means of the *External Event Manager*.
- *Temporal events* are subdivided into *instant*, *periodic*, and *interval events*. Interval events are particularly powerful, since they allow the binding of a time interval to another event (*anchor event*). For example, the expression "five minutes after the access to page X" represents a temporal event that is raised after the expiration of 5 minutes from the *anchor event* "access to page X". Temporal events are managed by the *Temporal Event Manager*, based on interrupts and the system clock.

The managers for external and temporal events are general in nature and easily reusable. The *Data Event Manager* is database-dependent[2]. The *Web Event Manager* requires a tight integration with the Web application.

Action Enactors. *Actions* correspond to modifications to the Web application or to executions of back-end operations. Typical adaptation actions are: adaptation of page contents, automatic navigation actions, adaptation/restructuring of the hypertext structure, adaptation of presentation properties, automatic invocation of operations or services. Adaptations are performed according to the user's profile or his/her context data.

While some actions can easily be implemented without any explicit support from the Web application (e.g. the adaptation of page contents may just require the setting of suitable page parameters when accessing the page), others may require a tighter integration into the application's runtime environment (e.g. the restructuring of the hypertext organization). The level of application support required for the implementation of the adaptivity actions thus heavily depends on the actual adaptivity requirements. However, application-specific actions can easily be integrated into the ECA-Web rule logic and do not require the extension of the syntax of the rule language (an example of the use of actions is shown in Figure 7).

[2] In our current implementation we support PostgreSQL. Modules for other database management systems are planned for future releases.

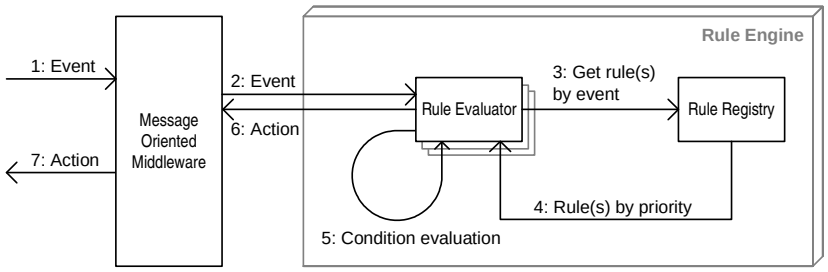

Fig. 3. The rule engine: internal rule execution logic.

As depicted in Figure 2, the execution of adaptivity actions is performed by means of three action enactors: *Web Action Enactor*, *External Action Enactor*, and *Data Action Enactor*. Web actions need to be provided by the application developer in terms of Java classes; they are performed by the *Web Action Enactor*, which is integrated into the application runtime environment, in order to guarantee access to the application logic. External actions are enacted through a dedicated Web service interface. Data actions are performed on the database that hosts the application's data source.

The enactor for external actions is general in nature and easily reusable, the *Data Action Enactor* is database-dependent, the *Web Action Enactor* is integrated with the Web application.

Rule Engine. In the architecture depicted in Figure 2, the *Rule Engine* is in charge of identifying the ECA-Web rules that correspond to captured events, of evaluating conditions, and of invoking action enactors – in case of conditions evaluating to true.

In the rule engine, a scalable, multithreaded *Rule Evaluator* evaluates conditions to determine whether the rule's action is to be performed or not, depending on the current state of the application. In ECA-Web, *conditions* consist of predicates over context data, application data, global session variables, and/or page parameters. For example, in the condition part of an ECA-Web rule it is possible to specify parametric queries over the application's data source, where parameters can be filled with values coming from session variables or page parameters.

The rule engine also includes a *Rule Registry* for the management of running, deployed ECA-Web rules. Deployed rules are loaded into the *Rule Registry*, a look-up table for the efficient retrieval of running rules, starting from captured events. The internal execution logic of a triggered rule is graphically summarized in Figure 3.

3.3 ECA-Web Rule Management

While the *Rule Registry* contains only deployed rules for execution, the *Rule Repository* offers support for the persistent storage of rules. For the management

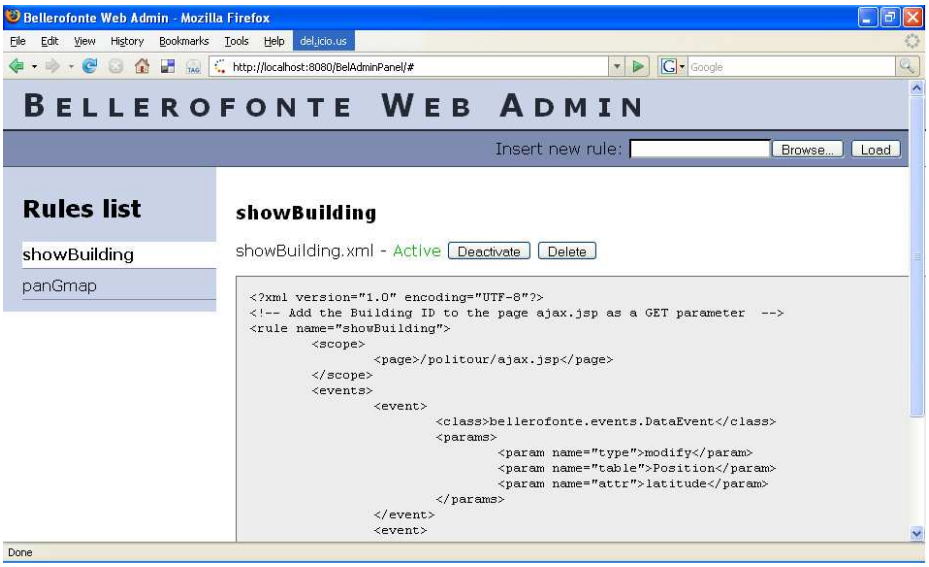

Fig. 4. The Web interface for the Rule Administration Panel.

of both *Rule Registry* and *Rule Repository*, we provide a *Rule Administration Panel* that allows designers to easily view, add, remove, activate, and deactivate rules. Figure 4 shows a screenshot of the *Rule Administration Panel*.

3.4 Deploying ECA-Web Rules

Activating or deploying an ECA-Web rule is not a trivial task and, depending on the rule specification, may require to set up a different number of modules. During the deployment of an ECA-Web rule, the XML representation of the rule is decomposed into its constituent parts, i.e. scope, events, conditions, action, and priority, which are then individually analyzed to configure the system. The scope is used to configure the *Web Event Manager* and the *Web Action Enactor*. The events are interpreted to configure the respective event managers and to set suitable triggers in the application's data source. The conditions are transformed into executable, parametric queries in the *Rule Registry*. The action specification and the rule's priority are as well fed into the *Rule Registry*. Each active rule in the system is thus represented by an instance in the *Rule Registry*, (possibly) by a set of data triggers in the database, and by a set of configurations of the event managers and the action enactors.

The registry allows the concurrent access by multiple *Rule Evaluators*. Priorities are taken into account in the action enactor modules, which select the action to be performed for the page under computation (the scope) from the queue of possible actions, based on rule priorities.

During the deployment of an ECA-Web rule, conflict resolution and termination analyses will be performed in line with the methods conceived and implemented for the Chimera-Exception language [17].

3.5 Enacting Adaptivity

External and *data* actions can be executed immediately upon reception of the respective instruction from the rule engine. The enaction of *Web* actions, which are characterized by adaptations visible on the user's browser, is possible only when a "request for adaptation" (a page request) comes from the browser. In fact, only in presence of an explicit page request, the Web application is actually in execution and, thus, capable to apply adaptations. This is due to the lack of suitable push mechanisms in the standard HTTP protocol.

In order to provide the application with active/reactive behaviors, in our previous works we therefore studied two possible solutions: (i) periodically *refreshing* the adaptive page currently viewed by the user [2], and (ii) periodically monitoring the execution context in the background (e.g. by means of suitable Rich Internet Application – RIA – technologies) and refreshing the adaptive page only in the case adaptivity actions are to be performed [7, 8]. Both mechanisms are compatible with the new rule-based architecture and enable the application to apply possible adaptivity actions that have been forwarded to the *Web Action Enactor* by the *Rule Engine*.

4 Case Study

In the context of the Italian research project MAIS[3] 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 through a PDA equipped with a GPS receiver for location sensing. User positioning is 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. The required adaptivity consists of (i) adapting page contents according to the user's position, and (ii) alerting the user of possible low connectivity conditions, based on the RSSI (Received Signal Strength Indicator) value of the wireless Internet connection. The alert consists in changing the background color of the displayed page.

The application has been designed with the WebML model, a visual notation for specifying the content, composition, and navigation features of hypertext applications [13]. In this paper we use the WebML notation for two distinct purposes: (i) to easily and intuitively describe the reference application, and (ii) to better highlight how the active rules introduced in the next section may take advantage from a formally defined, conceptual application model for the definition of expressive adaptivity rules. The approach we propose in this paper,

[3] http://www.mais-project.it

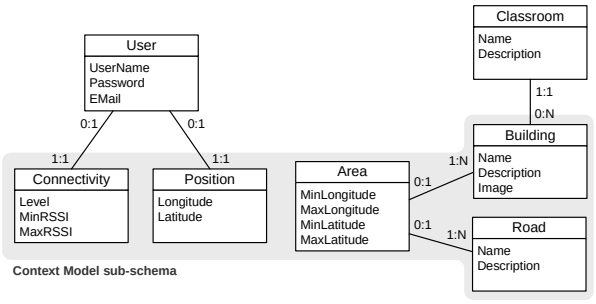

Fig. 5. ER data schema of the PoliTour application.

however, is not tightly coupled to WebML and can be used in the context of any modeling methodology upon suitable adaptation.

It is worth noting that the approach based on ECA-Web described in this paper is not to be considered an *alternative* solution to the conceptual design approaches so far proposed in the literature for Web application modeling. Rather, we believe that the best expressiveness and a good level of abstraction for the illustrated adaptivity specification language will be achieved by *complementing* the current modeling and design methods (such as WebML, Hera, OO-H or OOHDM). In fact, in this paper we hint at the specification of ECA-Web rules on top of WebML (both data and hypertext models), just like SQL triggers are defined on top of relational data models. This consideration is in line with the proposal by Garrigós et. al [6], who show how to apply their PRML rule language to several different conceptual Web application models.

The conceptual model of the application serves as terminological and structural reference model for the specification of adaptivity rules and, thus, allows application developers to keep the same level of abstraction and concepts already used for the design of the main application. In terms of WebML, for example, this could mean to restrict the scope of individual rules to specific hypertext elements like *content units*, *pages*, or *areas*, or to relate events to specific *links* or *units*. The same holds for actions, which could for example be applied to single *units* or even *attributes*.

4.1 Application Design with WebML

Figure 5 depicts a simplified version of the data schema underlying the PoliTour application, expressed in the Entity-Relationship (ER) notation. Five entities compose the context model, which is required in addition to the user identity to achieve the context-aware features of the application. The entities `Connectivity` and `Position` are directly connected to the entity `User`, as they represent context data which are individual for each user of the system. `Position` contains the latest GPS coordinates for each user, `Connectivity` contains a set of discrete connectivity levels that can be associated to users, based on their current RSSI. GPS coordinates and RSSI are sensed at the client side and periodically

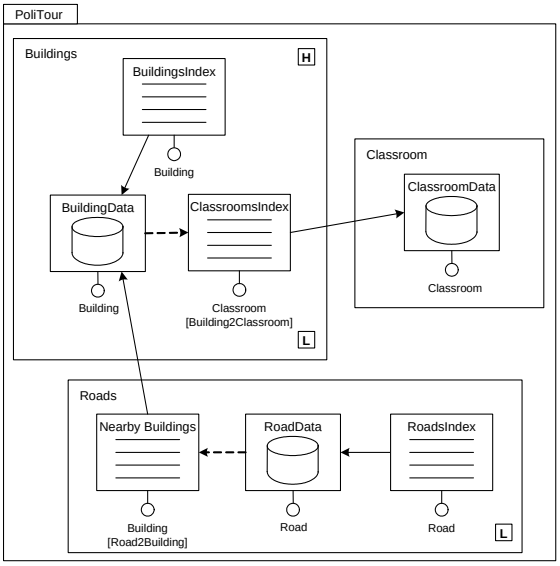

Fig. 6. Simplified hypertext model of the PoliTour application. `H` stands for `Home` page; `L` stands for `Landmark` page.

communicated to the application server in the background [7]. The entities `Area`, `Building`, and `Road` provide a logical abstraction of raw position data: buildings and roads are mapped onto a set of geographical areas inside the university campus, which enables the association of a user with the building or road he/she is located in, based on the GPS position. The entity `Classroom` is located outside the context model, as the application is not able to react to that kind of granularity, and the respective data is considered additional application content.

Figure 6 depicts the WebML-based hypertext schema of the PoliTour application defined on top of the ER schema shown in Figure 5. The application hypertext is composed of three pages: `Buildings`, `Roads`, and `Classroom`. Page `Buildings` shows a list of buildings (`BuildingsIndex` unit) the user can select from. By choosing one of the buildings, the respective details (`BuildingData` unit) and the list of classrooms (`ClassroomsIndex` unit) of the building is shown. If interested in, the user can select one of the building's classrooms and navigate to the `Classroom` page. Similarly, page `Roads` shows a list of roads for selection by the user. The details of selected roads are shown by the `RoadData` unit positioned in the middle of the page. The identifier of the selected road is further propagated to the `NearbyBuildings` unit, which shows the buildings adjacent to the road and allows the user to navigate to the `Buildings` page. The two pages `Buildings` and `Roads` are further tagged as *landmark* pages, meaning that they can be accessed through a global navigation menu. Page `Buildings` is also tagged as the *home* page of the application.

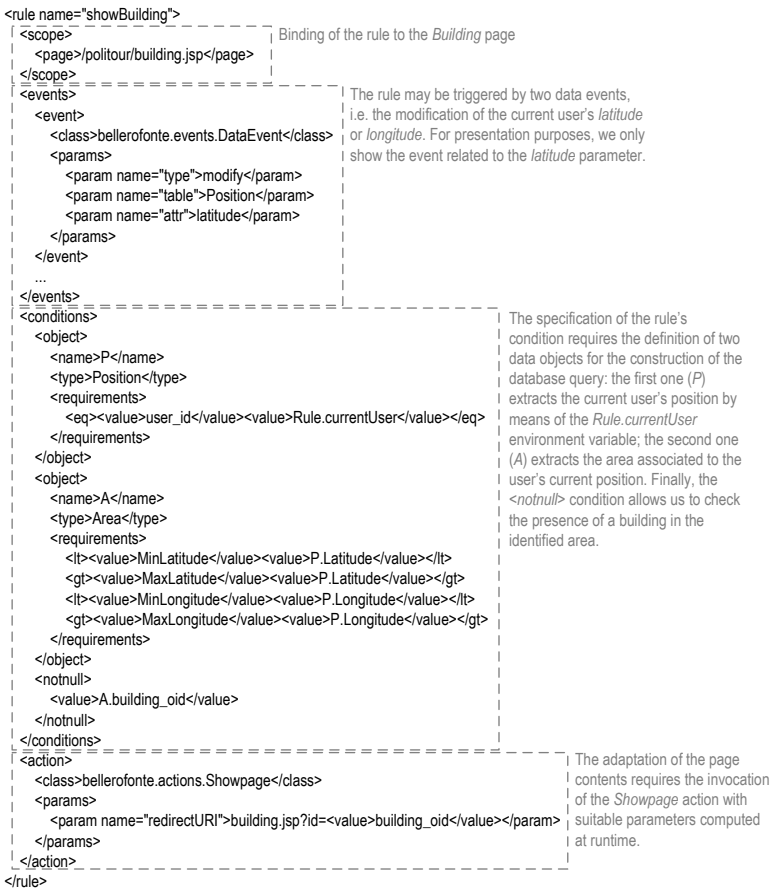

Fig. 7. The ECA-Web rule for checking the user's current position and updating page contents.

4.2 Defining an ECA-Web Rule

The full specification of the application's adaptivity requires several different ECA-Web rules to manage the adaptation of the contents in the pages `Buildings` and `Roads`, and to alert the user of low connectivity conditions. Figure 7 shows the ECA-Web rules that adapts the content of the page `Buildings` to the position of the user inside the university campus.

The scope of the rule binds the rule to the `Buildings` page. The triggering part of the rule consists of two data events, one monitoring modifications to the user's `longitude` parameter, one monitoring the user's `latitude` parameter. In the condition part of the rule we check whether there is a suitable building associated to the user's current position (`<notnull>` condition), in which case we enact the `Showpage` adaptivity action with new page parameters, suitably computed at runtime; otherwise, no action is performed. The condition evalua-

tion requires the extraction from the data source of two data items (`<object>`), namely the position of the current user and the area in which the user is located. The selection condition is enclosed within the `<requirements>` tag. In the action part of the rule we link the `bellerofonte.actions.Showpage`[4] Java class, which contains the necessary logic for the content adaptation action. The variable `building_oid` has been computed in the condition part of the rule and is here used to construct the URL query to be attached to the automatic page request that will cause the re-computation of the page and, thus, the adaptation of the shown content.

It is worth noting that the scope of the previous rule is limited to one specific hypertext page. There might be situations requiring a larger scope. For example, the rule for alerting users about low connectivity is characterized by a scope that spans all the application's pages; in terms of WebML, binding an ECA-Web rule to all pages means to set the scope of the rule to the site view, i.e. a model element (see site view `PoliTour` in Figure 6). The scope of the rule is specified as follows:

```
<scope>
  <siteview>PoliTour</siteview>
</scope>
```

As for the dynamic management of adaptivity rules, we could for example be interested in testing the two adaptivity features (location-aware contents and the low connectivity alert) independently. We would thus first only deploy the rule(s) necessary to update the contents of the `Buildings` and `Roads` pages and test their functionality without also enabling the alert. Then we could disable this set of rules and enable the rule for the alert and test it. If both tests are successful, we finally could enable both adaptivity features in parallel and test their concurrent execution.

5 Implementation

The proposed solution has been developed with scalability and efficiency in mind. The Web application and the rule engine are completely decoupled, and all communications are based on asynchronous message exchanges based on JMS (Java Message Service). The different modules of the proposed system can easily be distributed over several server machines. The overhead introduced into the Web application is reduced to a minimum and only consists of (i) forwarding Web events and (ii) executing adaptivity actions. These two activities in fact require access to the application logic. In fact, depending on the required adaptivity support, event mangers and action enactors may require different levels of customization by the Web application developer. The customization consists in the implementation of the application-specific events and of the actions that are to be supported by the adaptive application.

[4] *Bellerofonte* is the current code name of the rule engine project.

To perform our first experiments with ECA-Web and the rule engine, we have adapted the PoliTour application, which we already extensively tested when developing our model-driven approach to the design of context-aware Web applications [7]. As for now, our experiments with a limited number of rules have yielded promising results. Experimentations with larger numbers of active rules, different adaptive Web applications, and several users in parallel are planned.

Also, to really be able to take full advantage of the flexibility provided by the decoupled adaptivity rule management, a set of suitable adaptivity actions needs to be implemented. Our current implementation provides support for data actions and a limited set of Web actions (namely, `ShowPage` for adapting page contents, and `ChangeStyle` for adapting presentation style properties). Data actions are currently applied only to entities and attributes that are directly related to the user for which the action is being executed. Also, condition evaluation is automatically confined to those context entities and attributes that are related to the user for which the rule is being evaluated. We are already working on extending condition evaluation to any application data, coming from the data source as well as from page and session parameters.

In the context of WebML, the provision of a set of predefined adaptivity actions will lead to a library of adaptivity actions, possibly integrated into the WebML runtime environment. In the case of general Web applications, the rule engine can be used in the same fashion and with the same flexibility, provided that implementations of the required adaptivity actions are supplied.

6 Conclusions

We believe that the decoupled runtime management of adaptivity features represents the next step in the area of adaptive Web applications. In this paper we have therefore shown how to empower design methods for adaptivity with the flexibility provided by a decoupled environment for the execution and the administration of adaptivity rules. The development of Web applications in general is more and more based on fast and incremental deployments with multiple development cycles. The same consideration also holds for adaptive Web applications and their adaptivity requirements. Our approach allows us to abstract the adaptive behaviors, to extract them from the main application logic, and to provide a decoupled management support, finally enhancing the maintainability and evolvability of the overall application.

In our future work we shall focus on the extension of the ECA-Web language to fully take advantage of the concepts and notations that can be extracted from conceptual Web application models (e.g. from WebML models). We shall also investigate termination, complexity, and confluence issues, trying to apply Chimera-Exception's Termination Analysis Machine [17] to ECA-Web. Extensive experimentations are planned to further prove the advantages deriving from the decoupled approach.

References

1. Frasincar, F., Houben, G.J.: Hypermedia Presentation Adaptation on the Semantic Web. In: Proceedings of AH'02, Málaga, Spain, Springer (2002) 133–142
2. Ceri, S., Daniel, F., Matera, M., Facca, F.M.: Model-driven Development of Context-Aware Web Applications. ACM TOIT **7** (2007)
3. Schwabe, D., Guimaraes, R., Rossi, G.: Cohesive Design of Personalized Web Applications. IEEE Internet Computing **6** (2002) 34–43
4. Baumeister, H., Knapp, A., Koch, N., Zang, G.: Modeling Adaptivity with Aspects. In Lowe, D., Gaedke, M., eds.: Proceedings of ICWE'05, Sydney, Australia. Volume 3579 of LNCS, Springer-Verlag Berlin Heidelberg (2005) 406–416
5. Garrigós, I., Casteleyn, S., Gómez, J.: A Structured Approach to Personalize Websites Using the OO-H Personalization Framework. In: Web Technologies Research and Development - APWeb 2005. Volume 3399/2005 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg (2005) 695–706
6. Garrigós, I., Gómez, J., Barna, P., Houben, G.J.: A Reusable Personalization Model in Web Application Design. In: Proceedings of WISM'05, Sydney, Australia, University of Wollongong, School of IT and Computer Science (2005) 40–49
7. Ceri, S., Daniel, F., Facca, F.M., Matera, M.: Model-Driven Engineering of Active Context-Awareness. To appear in the World Wide Web Journal, Springer (2007)
8. Daniel, F., Matera, M., Pozzi, G.: Combining Conceptual Modeling and Active Rules for the Design of Adaptive Web Applications. In: Workshop Proceedings of ICWE'06, New York, NY, USA, ACM Press (2006) 10
9. De Bra, P., Aerts, A., Berden, B., de Lange, B., Rousseau, B., Santic, T., Smits, D., Stash, N.: AHA! The Adaptive Hypermedia Architecture. In: Proceedings of HYPERTEXT'03, (2003) 81–84
10. Kappel, G., Pröll, B., Retschitzegger, W., Schwinger, W.: Modelling Ubiquitous Web Applications - The WUML Approach. In: Revised Papers from the HUMACS, DASWIS, ECOMO, and DAMA on ER 2001 Workshops, London, UK, Springer-Verlag (2002) 183–197
11. Casteleyn, S., De Troyer, O., Brockmans, S.: Design time support for adaptive behavior in Web sites. In: Proceedings of SAC'03, New York, NY, USA, ACM Press (2003) 1222–1228
12. Troyer, O.D., Leune, C.J.: WSDM: A user centered design method for Web sites. Computer Networks **30** (1998) 85–94
13. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kauffmann (2002)
14. Alferes, J.J., Amador, R., May, W.: A general language for evolution and reactivity in the semantic web. In: Principles and Practice of Semantic Web Reasoning. Volume 3703 of LNCS, Springer Verlag (2005) 101–115
15. Bonifati, A., Braga, D., Campi, A., Ceri, S.: Active XQuery. In: Proceedings of ICDE?02, San Jose, California. (2002)
16. Bailey, J., Poulovassilis, A., Wood, P.T.: An Event-Condition-Action Language for XML. In: Proceedings of WWW?02, Hawaii. (2002) 486–495
17. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and implementation of exceptions in workflow management systems. ACM TODS **24** (1999) 405–451