Journal of Web Engineering, Vol. 7, No. 3 (2008) 179–199

MANAGING RUNTIME ADAPTIVITY THROUGH ACTIVE RULES: THE BELLEROFONTE FRAMEWORK

FLORIAN DANIEL, MARISTELLA MATERA, GIUSEPPE POZZI
Politecnico di Milano, Dipartimento di Elettronica e Informazione,
P.za L. da Vinci 32, 20133 Milano, Italy
{*daniel,matera,pozzi*}*@elet.polimi.it*

Received November 2, 2007
Revised March 10, 2008

Modern Web development is more and more moving towards the production of full-fledged, complex Web applications, possibly equipped with active and/or adaptive behaviors. On the one side, this evolution implies higher development costs and times; on the other side, 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 focus on the above problem, considering adaptive Web applications. We defined an Event-Condition-Action (ECA) rule-based approach aimed at facilitating the management and evolution of adaptive application features and we developed an engine, namely Bellerofonte, to process ECA rules. In our approach, we decouple the active logic (i.e. the adaptivity rules) from the execution of the actual application by means of a decoupled rule engine capturing events and autonomously enacting the adaptivity actions.

Keywords: Adaptive Web applications, Context-aware Web applications, Adaptivity, Context-awareness, ECA rules, ECA-Web

1 Introduction

Adaptability (the design-time adaptation of an application to user preferences and/or device characteristics [17]) and *adaptivity* (the run-time adaptation of an application to a user profile or a context model [17]) 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 the 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 [12], where adaptivity is based on a dynamically updated context model upon which the adaptive application is built.

Characteristic of the Web engineering discipline, the previous approaches to adaptability, adaptivity, and context-awareness 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 [17, 12], object-oriented approaches [22], aspect-oriented approaches [5], and rule-based paradigms [18, 19] 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 the coherence and readability of the resulting code. 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 [12, 11], we are convinced that the next step in this research area is to support a *dynamic* management of adaptivity features. We indeed believe that a decoupled paradigm for adaptivity rule specification, deployment, and administration can make rule management easier, especially after the implementation and deployment of the application. On the 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 [15] 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) at runtime. As envisioned above, with 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 related works on adaptivity in the Web. Section 3 provides an overview of the design concerns tackled in this paper. Section 4 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 5 discusses the prototype of an adaptive Web application supported by ECA-Web rules and shows the usage of the active rule language. Section 6 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. Section 7 discusses termination and confluence, which are relevant topics of any active system, for ECA-Web rules. Finally, Section 8 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 adaptations. In [17] 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 [7]. The UWA Consortium proposes WUML [21] for conceptual hypertext design. Adaptation requirements are expressed by means of OCL-based customization rules, referring to UML class or package elements. In [9] the authors present an extension of WSDM [23] 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/from a node), and links (adding/deleting links). In [5] 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 [12]. 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 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 features, 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 [1, 6, 2], ECA rules have been recently adopted to support adaptivity 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 [18] 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 [19], but its real potential for adaptivity management also at runtime remains unexplored.

Different transcoding solutions also adopt active rules for adapting Web pages. Most of them however focus on the presentation layer and provide mechanisms to transform HTML pages according to (possibly limited) device capabilities [20] or users' visual disabilities [24]. Moreover, they typically support only adaptability and modify Web pages in relation to a static set of user or device parameters. In [16] authors adopt the transcoding paradigm for the development of the Generic Adaptation Component (GAC). GAC provides a broad range of adaptation behaviors, especially supporting run time adaptivity. An RDF-based rule language is used for specifying both content adaptation and context data update rules. A collection of operations implementing these rules is provided. A notable feature, promoting portability, is that GAC can be integrated as a stand-alone module into the Web site architecture.

In line with the previous works, the approach we describe in this paper 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,[a] it is a self-sufficient language for the specification of adaptivity rules. The strength of our work is however the development of a decoupled environment for both the execution and administration of adaptivity rules, which allows the management of adaptivity features

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

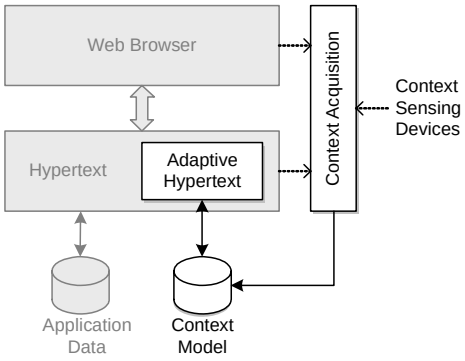

Fig. 1. Context data in adaptive Web applications. Gray shaded boxes correspond to conventional, non-adaptive parts; white boxes correspond to extensions required to support adaptivity; dashed arrows indicate the source of possible context data.

to be kept totally independent of the application execution. This aspect introduces several advantages in terms of maintainability and evolvability.

3 Design Concerns for Adaptive Web Applications

Developing adaptive Web applications typically requires some additional development steps compared to the development of traditional, non adaptive Web applications, as adaptive applications present some distinguishing characteristics that need to be taken into account. Figure 1 shows our interpretation of adaptive or context-aware Web applications: The overall application is composed of *adaptive* and *non-adaptive* parts. The non-adaptive hypertext leverages *application data* for the rendering of the hypertext pages; the adaptive hypertext also leverages a *context model*, which contains the execution context data that are at the basis of the adaptive behaviors of the application (e.g. user profile data or environmental context data). In order to always have an up-to-date picture of the application's context, a suitable *context acquisition* mechanism continuously updates the context model with fresh data.

According to this interpretation, enabling adaptivity requires suitable solutions addressing the following design concerns:

- *Context model definition and representation* in an application-accessible format. Context properties that are required to support the application's adaptivity features need to be identified and modeled as data objects that build up the context model.
- *Context model management*, consisting of:
  - *Context data acquisition* by means of acquisition of user data that are either explicitly specified by the user or implicitly captured during user interaction by tracking the user's navigations [10] or by means of measures of real-world, physical context properties, characterizing the usage environment. This activity is highly application-dependent and requires the design of suitable mechanisms and sensing infrastructures for capturing context data, possibly on both the client and the server side.

  - *Context data monitoring* to detect those variations in context data that trigger adaptivity [11]. Any variation may cause a context-triggered, adaptive behavior of the Web application. This activity abstracts the dynamics of context data and specifies which variations of the data require an adaptation of the application.

- *Hypertext adaptation.* If context monitoring detects a situation demanding for adaptation, then suitable adaptation operations need to be enacted, in order to translate the detected context state into visible effects or operations that augment the effectiveness and usability of the application.

So far, our work on adaptivity [12, 11] has been conceived in the context of a model-driven methodology for the development of Web applications, which is based on the adoption of the WebML conceptual model [13]. WebML proposes some modeling abstractions for data and hypertext design. Data design is based on the well-know Entity-Relationship (ER) model. Hypertext design, instead, exploits some original conceptual primitivies that are able to cover a large set of requirements arising in the design of Web applications.

According to the proposed approach for the design of adaptive Web applications, the specification of adaptivity actions is tightly coupled with the design of the hypertext, and consists in "attaching" suitable adaptive behaviors to those pages that are part of the adaptive hypertext. Actually, not only pages but also areas (i.e., clusters of pages) and entire hypertexts (*site views* in WebML terminology) may be tagged as adaptive and associated with a chain of WebML operations, representing the adaptivity logic associated to them and executed each time context monitoring (restricted to the currently viewed page) demands for an adaptation. The modeling of adaptive actions leverages existing WebML operation units, but also a set of newly introduced, adaptivity-specific units, especially related to the acquisition and management of context data. For more details on such new units the reader is deferred to [12, 11, 14, 10].

The experience of extending WebML for adaptivity allowed us to identify advantages and drawbacks of the tight integration of application and adaptivity logics. Despite the multiple advantages offered at design time by the adoption of a conceptual model, we in particular observed that the management and evolution of the adaptivity rules after the application deployment is cumbersome, since it requires the modification of the design schema and a new deployment of the application. This motivation led us to develop a decoupled paradigm for the specification of adaptivity rules, which addresses the previous design concerns from a slightly different perspective. In the following, we describe the result of this effort, the Bellerofonte framework for the dynamic management of adaptivity features.

4 Enabling Dynamic Adaptivity Management

The runtime management of an application's adaptive behaviors is articulated in two complementary features: a *design* component (the ECA-Web language) and a *runtime* component (the rule execution environment). The design component enables the specification of adaptive behaviors, while the runtime component supports the execution and dynamic management of the specified behaviors in Bellerofonte.

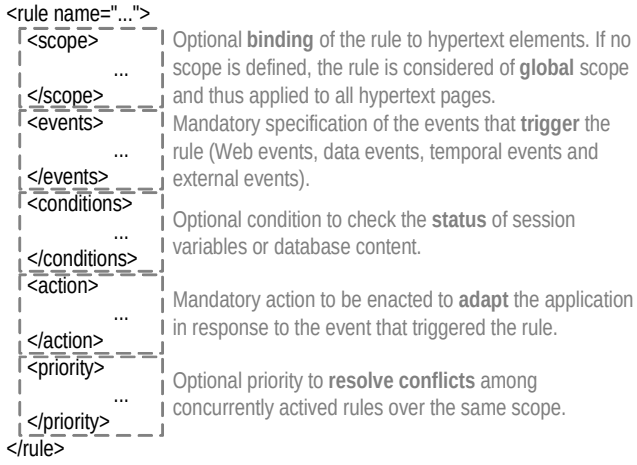

Fig. 2. Structure of ECA-Web rules.

4.1 The ECA-*Web Language*

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 [8]. ECA-Web is an evolution of the Chimera-Web language we already proposed in [15], 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 2. A typical ECA-Web rule is composed of five parts: scope, events, conditions, action, and priority.

4.1.1 Scope

The *scope* defines the binding of the rule with individual hypertext elements (e.g. pages, links, contents inside pages). In line with the classification of the application's pages into adaptive and non-adaptive as described in Figure 1, the definition of a suitable scope for adaptivity rules allows one to couple the respective adaptivity logic with individual hypertext elements. If the definition of the scope is omitted, the rule has a *global* scope, i.e., the rule is applied to all the pages of the application.

When specified for a rule, the definition of the `<scope>` element requires the indication of the hypertext elements the rule must be applied to. For example, the following XML code fragment specifies a scope where a `<page>` element contains the name of the page (`PageX`) that is affected by the rule.

```
<scope>
  <page>PageX</page>
</scope>
```

The scope can bind a rule also to multiple pages; as also shown in the case study description (see Section 5), this is achieved by considering page containers (e.g., areas or entire hypertext structures) as scope elements, but the scope may also be specified by simply listing the pages that are part of the scope.[b] At the current stage of Bellerofonte, the finest granularity level

[b] The scope elements depend on the abstractions provided by the adopted modeling language.

is however the page, since it is the lowest-level hypertext element that can be addressed by adaptivity actions with conventional Web technologies. We are however planning some extensions, based on Rich Internet Applications - RIA - technologies, to be able to address even single content chunks inside pages, thus providing a finer level of granularity for the rules.

4.1.2 Events

By means of *events* we specify how the rule is triggered in response to user navigations or changes in the underlying context model. ECA-Web provides support for the following kinds of events:

- *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.
- *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).
- *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 (Really Simple Sindacation).
- *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".

The following code lines exemplify the definition of a data event:

```
<events>
  <event>
    <class>bellerofonte.events.DataEvent</class>
    <params>
      <param name="type">modify</param>
      <param name="table">Position</param>
      <param name="attr">latitude</param>
    </params>
  </event>
</events>
```

Events are specified by referencing their class names in the Bellerofonte framework (see element `<class>`). Data events further require the definition of three parameters in input (by means of the `<params>` and `<param>` elements): the *type* of data event, i.e., create, modify, or delete, and the *table* and the *attr*(ibute) of the context model to be monitored. Web events, external events, and temporal events are specified analogously; a complete discussion of the event specification syntax would however be out of the scope of this paper.

The trigger of a rule may also consist of more than one event. In this case, the `<events>` element will contain one `<event>` element for each basic event. Multiple events follow a

disjunctive execution logic, i.e., in order for the respective rule to be triggered, it suffices that one of the declared events is fired. This policy is inherited from the Chimera-Exception language [8].

4.1.3 Conditions

The *condition* part evaluates the state of user-specific application data and decides whether a rule's action is to be executed or not in response to the triggering of the rule's event. Specifying a condition corresponds to the specification of a query over a user's context, profile, or session data. For example, in the condition part of an ECA-Web rule we can specify parametric queries over the application's data source, where parameters can be filled by values coming from session variables (referring to a specific user) or to page parameters.

By means of the shortcut `Rule.VarName` it is possible to directly access the user's personal session variables (e.g. `VarName`), while the access to database content is supported via the `<object>` element, which allows the definition of an object structure over entities (`<table>` element) in the underlying database. The object specification supports the use of a dot notation for accessing the values of the attributes of the object. Note that if an object definition returns more than one instance, in our current implementation only the first item is selected to populate the object structure. The following code snippet exemplifies the definition of an object `P` over a data entity `Position`:

```
<conditions>
  <object>
    <name>P</name>
    <table>Position</table>
    <requirements>
      <eq><value>P.user_id</value><value>Rule.currentUser</value></eq>
    </requirements>
  </object>
  <notnull>
    <value>P.latitude</value>
  </notnull>
</conditions>
```

The `<requirements>` construct allows one to define a selector condition over the entity, which is based on a few operators: equal to `<eq>`, less than `<lt>`, greater than `<gt>`, less than or equal to `<leq>`, greater than or equal to `<geq>`, not null `<notnull>`, different from `<diff>`. In the code lines above, the selector condition selects the object with `user_id` equal to `Rule.currentUser` (the id of the current user).

The only definition of objects does not yet fully specify a condition definition. A conditional statement over one of the object is required.This final conditional statement represents the actual value of the condition. For instance, the code lines above check whether the value of `P.latitude` is not null.

It is worth noting that in ECA-Web time-based conditions are expressed as time events, in the form of "an amount of time expired since the occurrence of an anchor event": as an example, a temporal event may allow one to express the condition "if 5 minutes elapsed since the page X has been visited". As for more complex conditions, further details can be found in [8, 10].

4.1.4 Actions

The *action* specifies the adaptation action that is to be performed in response to a triggered event and a true condition. Typical adaptive actions in the Bellerofonte framework are:

1. Adaptation of visualized *contents* or delivered *services*, for example to take into account the preferences specified in a user profile or the properties of the current context of use (e.g., geographical location, environmental conditions, system status, etc.).

2. Automatic *navigation* actions toward another page of the same application, which is better suited to the current context conditions, for example to enable (or disable) functions and contents depending on the current situation of use.

3. Adaptation of the whole *hypertext structure* for supporting coarse-grained adaptation requirements, for example due to changes of the user's device, role or activity within a multi-channel, mobile environment.

4. Adaptation of *presentation properties*, in order to provide more fine-grained adjustments of the application's presentation layer.

5. Modifications to the underlying *database content*, e.g. to update a counter variables.

6. Execution of *back-end operations*, e.g. to enact external application logics.

Similarly to events, also the definition of the action of an ECA-Web rule requires the referencing of the respective class in the Bellerofonte framework, and the passing of suitable parameter values, as exemplified in the following code lines that redirect the user to another page of the application:

```
<action>
  <class>bellerofonte.actions.Showpage</class>
  <params>
    <param name="redirectURI">NewPageName</param>
  </params>
</action>
```

We observe that the interpretation of actions in ECA-Web distinguishes the rule language for adaptive Web applications from the original rule language for exception handling in workflow management systems, Chimera-Exception. In fact, the set of available actions in ECA-Web extends the original set by adding actions such as the above mentioned ones, while ECA-Web does not include pure exception handling actions.

4.1.5 Priorities

The *priority* defines an execution order for rules that are concurrently activated over the same scope. If no priority is specified, a default priority value is assigned. The highest priority is represented by 0 (zero), and priorities decrease with the growing of the priority value (up to a maximum value of 255). Priorities are specified as follows:

```
<priority>10</priority>
```

In Section 7 we shall show how a sensible use of priorities allows the application designer to control the execution order of multiple concurrently activated rules.

4.2 The Integrated Runtime Architecture

The execution of ECA-Web rules demands for a proper runtime support, which is offered by the Bellerofonte framework. Figure 3 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

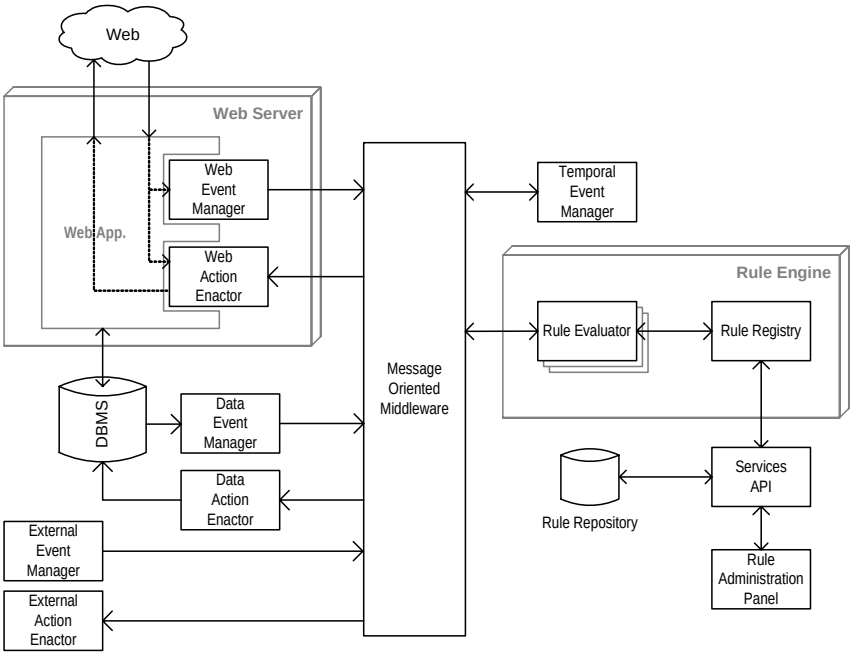

Fig. 3. Functional architecture of the integrated execution environment for adaptive Web applications.

the single modules are achieved through asynchronous message exchanges (*Message-Oriented Middleware*).

4.2.1 Event Managers

Each type of ECA-Web event is supported by a suitable event manager [15]:

- *Data events* are managed by the *Data Event Manager*, which runs on top of the application's data source.
- *Web events* are raised 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* are captured by means of the *External Event Manager*. When an external event occurs, the name of the triggering event and suitable parameters are forwarded to the rule engine.
- *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.[c] The *Web Event Manager* requires a tight integration with the Web application.

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

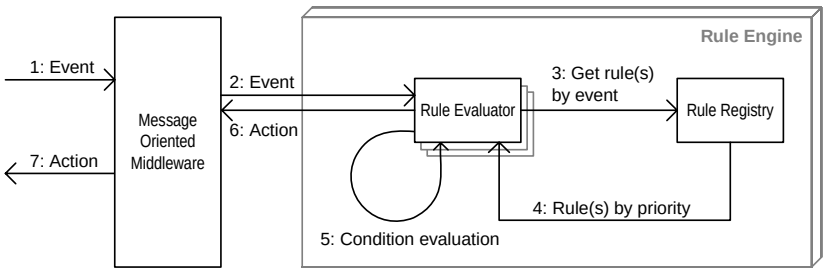

Fig. 4. The rule engine: internal rule execution logic.

4.2.2 Action Enactors

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 8).

As depicted in Figure 3, 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 as 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.

4.2.3 Rule Engine

In the architecture depicted in Figure 3, 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.

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 4.

4.3 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 of both *Rule Registry* and *Rule Repository*, we provide a *Rule Administration Panel* that allows designers to easily

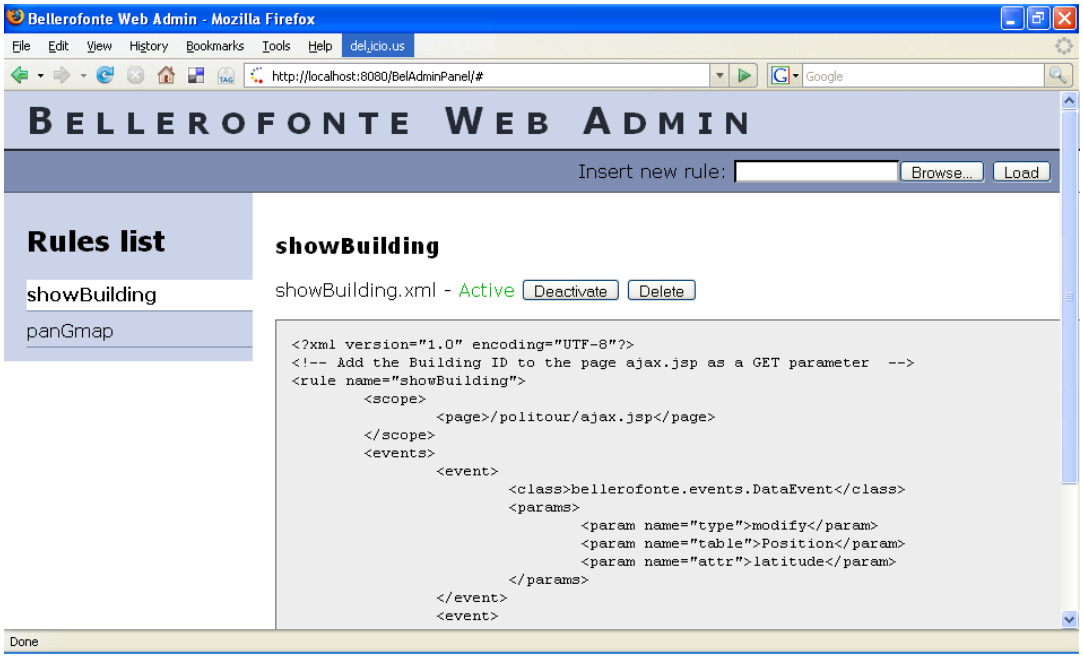

Fig. 5. The Web interface for the Rule Administration Panel of Bellerofonte.

view, add, remove, activate, and deactivate rules. Figure 5 shows a screenshot of the *Rule Administration Panel*.

4.4 Deploying ECA*-Web Rules*

Within the Bellerofonte framework, 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 provides 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 pages under computation (the scope) from the queue of possible actions, based on rule priorities.

As better illustrated in Section 7, 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 [8].

4.5 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 have studied two possible solutions: (i) periodically *refreshing* the adaptive page currently viewed by the user [12], 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 [11, 15]. 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*.

5 Case Study

In the context of the Italian research project MAIS,[d] 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 WebML. We observe that 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, however, is not tightly coupled to WebML and can be used in the context of any modeling methodology upon suitable adaptation.

We also observe that the approach based on ECA-Web rules 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 [19], 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 models for the specification of adaptivity rules and, thus, allows application developers to keep

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

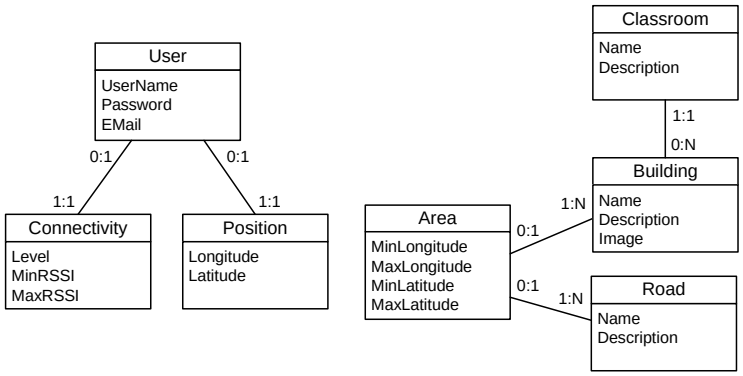

Fig. 6. ER data schema of the *PoliTour* application.

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*.

5.1 Application Design with WebML

Figure 6 depicts a simplified version of the data schema underlying the *PoliTour* application, expressed in the ER notation. All entities represent some context properties that underly the adaptive features of the application and are therefore part of the so-called *context model* [e].

The entity `User` stores data characterizing individual application users. 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 communicated to the application server in the background [11]. 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 one to associate a user with the building or road he/she is located in, based on the GPS position. The entity `Classroom` may be considered additional application data, as the application is not able to react to that kind of granularity (the GPS receiver works outdoors, only).

Figure 7 depicts the WebML-based hypertext schema of the *PoliTour* application defined on top of the ER schema shown in Figure 6. 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 are 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

[e] As can be noticed, we use an opportunistic model, which is specific to the particular application domain. Bellerofronte does not adopt a general context model. Also, the definition of a universal context model is not discussed in this paper and is out of the scope of our research.

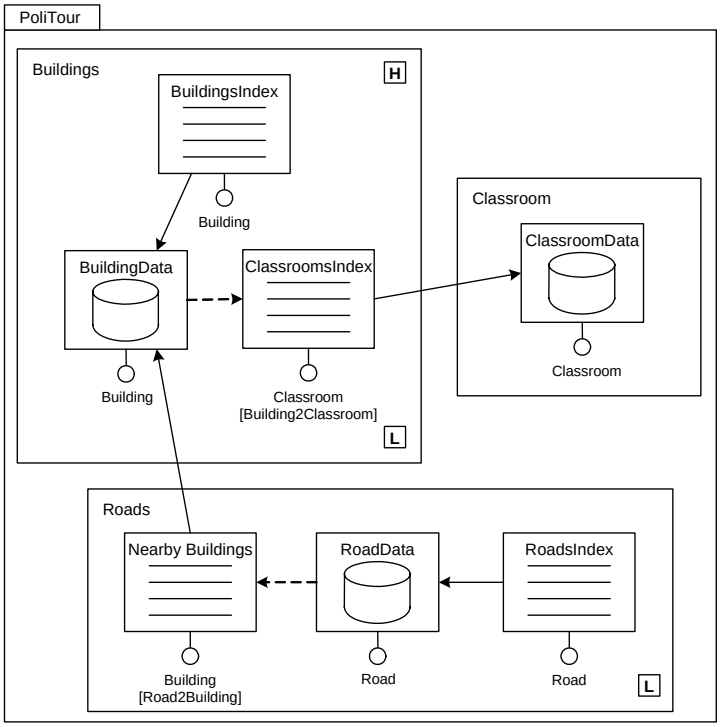

Fig. 7. Simplified hypertext model of the *PoliTour* application. `H` stands for `Home` page; `L` stands for `Landmark` page.

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.

5.2 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 8 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 evaluation requires to extract from the data source 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` Java class, which contains the necessary logic for the content adaptation action. The variable `building_oid` has been computed in

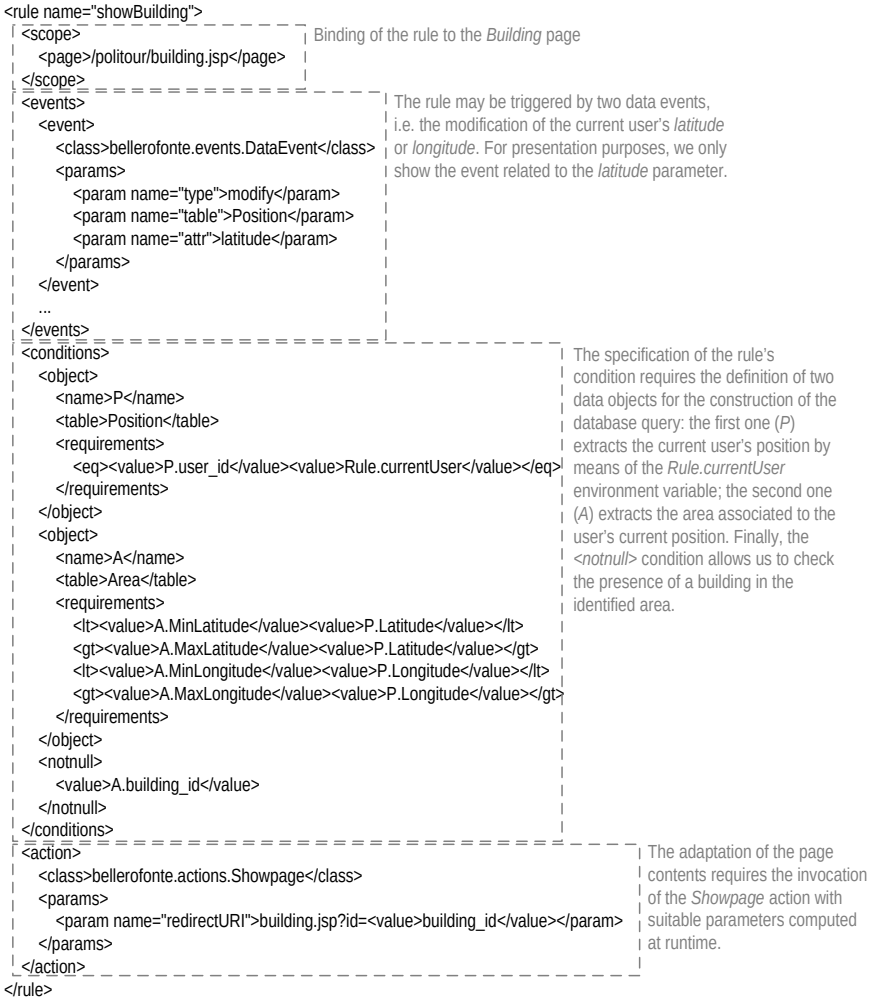

Fig. 8. The ECA-Web rule for checking the user's current position and updating page contents.

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.

We observe 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 7). 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.

6 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. 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.

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 [11]. 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 still used in the same fashion and with the same flexibility, provided that the language is adapted to take into account the primitives underlying the application design (e.g., "slices" instead of "pages") and that the implementations of the required Web event managers and Web action enactors are supplied.

7 Termination and Confluence

When dealing with active software components, the designer has to pay great attention to the characteristics of the entire set of resulting active behaviors. In particular, the designer must study the *termination* and *confluence* properties of the set of active behaviors, in order to identify possible cyclic behaviors or non-deterministic orderings of active behaviors, respectively [8].

7.1 Termination Analysis

Termination takes place when active rules do not trigger each other indefinitely. This, hopefully, means that rule processing eventually terminates or, alternatively, that rule execution never yields *infinite loops*. It is possible to distinguish three different kinds of typically undesired cyclic behaviors:

- *Self-triggering:* one rule triggers itself indefinitely. As an example, consider a rule R_1 triggered by event e_1 and executing action a_1. Self-triggering occurs if a_1 triggers e_1.
- *Cross-triggering:* two rules trigger each other indefinitely. Imagine, for example, to have two rules, R_1 and R_2, where R_1 is triggered by e_1 and executes a_1, and R_2 is triggered by e_2 and executes a_2. Cross-triggering occurs if a_1 triggers e_2 and a_2 triggers e_1.
- *Cascaded triggering:* multiple rules trigger each other indefinitely in a cascaded fashion. Consider, for instance, R_i with $0 \leq i \leq n$, where a_i triggers e_{i+1} for $0 \leq i < n$ and a_n triggers e_0: cascaded triggering occurs.

Termination is a well studied topic in the field of active databases, mainly focused on rules triggered by data events and performing actions that change the state of the database. From a strict database point of view, in Bellerofonte we assume that rules execute as ACID transactions upon the same database and their serializability is guaranteed by the underlying database server. However, ECA-Web rules extend the strict area of the pure database field, as actions are not limited to database changes only but also include Web actions or external actions.

As shown in [8], the analysis of termination is not a simple task, as it requires to detect all those situations that may lead to the self-triggering, cross-triggering, or cascaded triggering of active rules. For the analysis of the termination properties of ECA-Web rules, we thus introduce a Termination Analysis Machine (TAM), similarly to [8], which allows us to discover any possible non-termination at rule compilation time, i.e., when an ECA-Web rule is deployed, and to generate appropriate warning messages for the developer. TAM limits its analysis to the rules defined for single users of one single application. At present, in fact, we limit the data actions of ECA-Web rules to each user's personal profile data, and we do not allow two users to share personal profile data. Web actions are limited by construction to single user sessions only, while external actions are outside the control of the application.

For the detection of possible non-terminating behaviors, each time a new rule is deployed for a specific user, TAM builds up an activation tree of the user's active rules. Every node of the tree relates to one trigger and is identified by the unique trigger name. Any action of any trigger is depicted by one arc inside the tree, regardless of the condition of the trigger itself. In fact, condition evaluation takes place at run-time, and the returned value cannot be estimated in advance. If the action can, in turn, activate an event, the arc is drawn from the trigger whose action is considered to the trigger whose event is activated by that action. The TAM then checks whether any cycle can be individuated inside the graph: each detected cycle may yield a non-termination, depending on the evaluation of the condition part of the trigger.

Hence, cycles represent only potential situations of non-termination. Oftentimes, cyclic triggering graphs are acceptable, provided that each cycle is analyzed in order to test whether termination is in danger. In this regard, the literature considers for example semantic rule analysis methods (e.g. [3, 4]), which can be extended to our context, even if several practical difficulties may arise. From a practical point o view, however, it is worth noting that in our

approach to adaptive Web applications, the user always has the possibility to simply stop the active behavior of the application. Rule evaluation occurs at discrete time intervals only (after each polling interval), which always guarantees the user a higher priority and prevents the complete loss of control.

7.2 Confluence

Confluence of active behaviors means that the final effect of the processing of multiple concurrently triggered rules is independent of the *ordering* by which rules are triggered and executed.

In general, whenever a language enables the user to perform a declarative selection of a set of elements, followed by the individual management of each element in the set, confluence is not guaranteed. As an example, this situation typically occurs in most SQL triggers and procedures of conventional database applications, which, thus, inherently implement non-confluent behaviors. The same consideration applies to ECA-Web: each rule is intrinsically *nondeterministic*, because it associates a set-oriented, declarative condition with a tuple-oriented imperative action. There is no language construct for imposing a rule-internal order on the bindings that are selected by the condition.

The complete analysis of confluence in ECA-Web requires to study also the case where two (or more) rules react to the same event. For instance, we might have rule R_1 which is triggered by event e_1, checks condition c_1, and possibly executes action a_1, which redirects navigation to page p_1; we might also have rule R_2 which is triggered by event e_1, checks condition c_2, and possibly executes action a_2, which redirects navigation to page p_2. The following two situations may happen:

- c_1 and c_2 are *mutually exclusive*: If $c_1 = \overline{c_2}$, we can say for sure which is the finally displayed page, according to the result of the condition evaluation;
- c_1 and c_2 are *not mutually exclusive*: If, instead, $c_1 \neq \overline{c_2}$, we cannot say for sure if the page finally displayed is p_1 or p_2.

In order to enable the developer to clearly state which is the desired order of rules (and of actions), ECA-Web adopts the following policy, based on the assignment of suitable *priorities* to rules (see Section 4.1.5):

- If two or more rules with the *same priority* can be triggered by the same event, their conditions must be mutually exclusive. Otherwise, the rule compiler generates an error message.
- If two or more rules with *different priorities* can be triggered by the same event, their conditions not necessarily need to be mutually exclusive. However, if rules with different priority are triggered concurrently and their respective conditions are evaluated true, only the rule with highest priority (i.e. the one with the lowest `<priority>` value) is processed, the one with lower priority is discarded and cancelled.

The described policy is a major difference with respect to [8] and, in general, to traditional active database systems, where all the triggers, also low priority ones, are always executed and never cancelled by higher priority triggers. However, in the specific context of adaptive Web applications, where actions may imply the re-direction of the user to another page or the re-arrangement of the page layout, we have opted for a simple rule execution logic, in order to keep the resulting adaptive behaviors intuitive to users. Also, executing a rule with low

priority immediately after the execution of a rule with higher priority may lead to obsolete or wrong adaptive behaviors, as the execution of the high priority rule in general also changes the actual execution context. A *sensible* design of ECA-Web rules thus guarantees comprehensible application behaviors.

8 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). Extensive experimentations are planned to further prove the advantages deriving from the decoupled approach.

Acknowledgements

We thank the Master students Alessandro Morandi and Matteo Mortari for their valuable support in implementing the prototype of Bellerofonte. This research has partially been funded by the European Social Fund.

References

1. José Júlio Alferes, Ricardo Amador, and Wolfgang May. A General Language for Evolution and Reactivity in the Semantic Web. In François Fages and Sylvain Soliman, editors, *PPSWR*, volume 3703 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2005.
2. James Bailey, Alexandra Poulovassilis, and Peter T. Wood. An Event-condition-action Language for XML. In *WWW*, pages 486–495, 2002.
3. Elena Baralis, Stefano Ceri, and Stefano Paraboschi. Run-time Detection of Non-Terminating Active Rule Systems. In Tok Wang Ling, Alberto O. Mendelzon, and Laurent Vieille, editors, *DOOD*, volume 1013 of *Lecture Notes in Computer Science*, pages 38–54. Springer, 1995.
4. Elena Baralis, Stefano Ceri, and Stefano Paraboschi. Compile-Time and Runtime Analysis of Active Behaviors. *IEEE Trans. Knowl. Data Eng.*, 10(3):353–370, 1998.
5. Hubert Baumeister, Alexander Knapp, Nora Koch, and Gefei Zhang. Modelling Adaptivity with Aspects. In David Lowe and Martin Gaedke, editors, *ICWE*, volume 3579 of *Lecture Notes in Computer Science*, pages 406–416. Springer, 2005.
6. Angela Bonifati, Daniele Braga, Alessandro Campi, and Stefano Ceri. Active XQuery. In *ICDE*, pages 403–412, 2002.
7. Paul De Bra, A. T. M. Aerts, Bart Berden, Barend de Lange, Brendan Rousseau, Tomi Santic, David Smits, and Natalia Stash. AHA! The Adaptive Hypermedia Architecture. In *Hypertext*, pages 81–84. ACM, 2003.
8. Fabio Casati, Stefano Ceri, Stefano Paraboschi, and Giuseppe Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Trans. Database Syst.*, 24(3):405–451, 1999.
9. Sven Casteleyn, Olga De Troyer, and Saar Brockmans. Design Time Support for Adaptive Behavior in Web Sites. In *SAC*, pages 1222–1228. ACM, 2003.

10. Stefano Ceri, Florian Daniel, and Federico Michele Facca. Modeling Web Applications Reacting to User Behaviors. *Computer Networks*, 50(10):1533–1546, 2006.
11. Stefano Ceri, Florian Daniel, Federico Michele Facca, and Maristella Matera. Model-Driven Engineering of Active Context-Awareness. *World Wide Web Journal*, 10(4):387–413, 2007.
12. Stefano Ceri, Florian Daniel, Maristella Matera, and Federico Michele Facca. Model-driven Development of Context-aware Web Applications. *ACM Trans. Internet Techn.*, 7(1), 2007.
13. Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
14. Florian Daniel. *Model-Driven Design of Context-Aware Web Applications*. Ph.D. Thesis, Politecnico di Milano, 2007.
15. Florian Daniel, Maristella Matera, and Giuseppe Pozzi. Combining Conceptual Modeling and Active Rules for the Design of Adaptive Web Applications. In Nora Koch and Luis Olsina, editors, *ICWE '06: Workshop proceedings of the sixth international conference on Web engineering*, page 10, New York, NY, USA, 2006. ACM Press.
16. Zoltán Fiala and Geert-Jan Houben. A generic transcoding tool for making web applications adaptive. In Orlando Belo, Johann Eder, João Falcão e Cunha, and Oscar Pastor, editors, *CAiSE Short Paper Proceedings*, volume 161 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
17. Flavius Frasincar and Geert-Jan Houben. Hypermedia Presentation Adaptation on the Semantic Web. In Paul De Bra, Peter Brusilovsky, and Ricardo Conejo, editors, *AH*, volume 2347 of *Lecture Notes in Computer Science*, pages 133–142. Springer, 2002.
18. Irene Garrigós, Sven Casteleyn, and Jaime Gómez. A Structured Approach to Personalize Websites Using the OO-H Personalization Framework. In Yanchun Zhang, Katsumi Tanaka, Jeffrey Xu Yu, Shan Wang, and Minglu Li, editors, *APWeb*, volume 3399 of *Lecture Notes in Computer Science*, pages 695–706. Springer, 2005.
19. Irene Garrigós, Jaime Gómez, Peter Barna, and Geert-Jan Houben. A Reusable Personalization Model in Web Application Design. In *Proceedings of ICWE 2005 Workshop on Web Information Systems Modelling (WISM2005)*, 2005.
20. Masahiro Hori, Goh Kondoh, Kouichi Ono, Shin'ichi Hirose, and Sandeep K. Singhal. Annotation-based Web Content Transcoding. *Computer Networks*, 33(1-6):197–211, 2000.
21. Gerti Kappel, Birgit Pröll, Werner Retschitzegger, and Wieland Schwinger. Modelling Ubiquitous Web Applications - The WUML Approach. In Hiroshi Arisawa, Yahiko Kambayashi, Vijay Kumar, Heinrich C. Mayr, and Ingrid Hunt, editors, *ER (Workshops)*, volume 2465 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2001.
22. Daniel Schwabe, Robson Guimarães, and Gustavo Rossi. Cohesive Design of Personalized Web Applications. *IEEE Internet Computing*, 6(2):34–43, 2002.
23. Olga De Troyer and C. J. Leune. WSDM: A User Centered Design Method for Web Sites. *Computer Networks*, 30(1-7):85–94, 1998.
24. Yeliz Yesilada, Simon Harper, Carole A. Goble, and Robert Stevens. Screen readers cannot see: Ontology based semantic annotation for visually impaired web travellers. In Nora Koch, Piero Fraternali, and Martin Wirsing, editors, *ICWE*, volume 3140 of *Lecture Notes in Computer Science*, pages 445–458. Springer, 2004.