Model-driven Engineering of Active Context-Awareness

Stefano Ceri Florian Daniel Federico M. Facca Maristella Matera

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

Abstract. More and more Web users ask for contents and services highly tailored to their particular contexts of use. Especially due to the increasing affordability of new and powerful mobile communication devices, they also appreciate the availability of ubiquitous access, independent from the device actually in use. Due to such premises, traditional software design methods need to be extended, and new issues and requirements need to be addressed for supporting context-aware access to services and applications.

In this paper we propose a model-driven approach towards adaptive, context-aware Web applications, accompanied by a general-purpose execution framework enabling *active context-awareness*. Whereas conventional adaptive hypermedia systems address the problem of adapting HTML pages in response to user-generated requests, in this work we especially stress the importance of user-independent, context-triggered adaptivity actions. This finally leads us to interpret the context as an *active* actor, operating independently from users during their navigations.

1 Introduction

Current advances in communication and network technologies are changing the way people interact with Web applications. They provide users with different types of mobile devices for accessing – at any time, from anywhere, and with any media – services and contents customized to the users' preferences and usage environments. Content personalization has already demonstrated its benefits for both users and content providers and has been commonly recognized as fundamental factor for augmenting the overall effectiveness of applications. Going one step further, context-awareness can be interpreted as natural evolution of personalization, addressing not only the user's identity and preferences, but also the environment that hosts users, applications, and their interaction, i.e., the context. Context-awareness, hence, aims at enhancing the application usefulness by taking into account a wide range of context properties.

Although many definitions of *context* are given by enumerating examples or by listing synonyms[1], inspired by the work of Dey and Abowd [17], we define *context* as *any information that can be used to characterize the interaction of a user with a software system (and vice-versa), as well as the environment where such interaction occurs.* This definition not only concentrates on interaction and environment properties, but also includes the user and software system themselves. We further define a system as *context-aware*, if *it uses context either for delivering content, or for performing system adaptations, or for doing both.*

In this paper we discuss some conceptual modeling facilities as well as technological support for capturing the peculiarities of context-aware behaviors in Web applications. An example of context-awareness in Web applications is the automatic update of Web page contents based on the user's position. This behavior can be very useful in some classes of mobile Web applications that are required to provide contents and services depending on the position of the user.

In order to cope with context-awareness, we have defined a model-driven approach that extends a well-known conceptual modeling language for Web applications, WebML (Web Modeling Language) [9]. The proposed methodology [6, 8, 38] offers the advantage of fully covering the design and the development of context-aware Web applications through automatic code generation, based on a consolidated CASE tool for Web application modeling [41]. In this paper, we specifically concentrate on *active, context-aware behaviors*, namely *context-triggered* adaptivity actions, which enable the system to automatically take the initiative of adapting to the current context of use, every time a significant context variation is detected and independently from users' requests.

Several adaptive Web systems (for example [18] or [20]) address the problem of adapting HTML pages in response to user-generated requests. In the domain of the Web, where the HTTP protocol imposes a strict pull paradigm to all communications, the most common solution consists in adapting pages only when explicitly requested or by periodically refreshing pages, thus polling adaptivity. Based on our previous experiences [6, 8, 38], neither of these two mechanisms results to be adequate for the special requirement of active, context-awareness. In this paper we therefore refine our previous work and introduce a model-driven approach to the design of context-aware Web applications, complemented with a *context monitor*, operating autonomously and transparently in the background to provide suitable active support. This finally leads to interpret context as "first class actor", operating independently from users on the same hypertext the users navigate. This is the main difference of the proposed approach with respect to other interpretations of adaptive hypertexts.

This paper is organized as follows. Section 2 characterizes context-aware Web applications by means of requirements and capabilities of such class of applications. Section 3 concentrates on modeling context data, and Section 4 extends WebML [9] towards context-aware Web applications. Section 5 introduces suit-

[1] In this paper, we will also speak about *adaptivity* and *adaptation*; *adaptability* intended as design time adaptation to device characteristics or user preferences is already natively supported by the adopted conceptual modeling language, WebML.

able runtime support enabling active, autonomous adaptivity, while Section 6 shows how the respective application code can be generated automatically. To exemplify the development of active, context-aware Web applications by means of the introduced instruments, Section 7 describes a case study. Section 8 discusses some related work, and Section 9 finally draws our conclusions and outlines our current and future work.

2 A Conceptual View over Context-Aware Web Applications

The discussion of context-awareness in Web applications first of all demands for a precise definition of the term *Web application*, so as to ground the proposed ideas and solutions on a technological context. We concentrate on the "classical" three-tier architecture of Web applications, consisting of Web browser for visualization (thin client with (X)HTML and CSS) and Web application server and database back-end for business logic and data management. Although in Section 5 we will make use of client-side scripting to enhance the user experience of adaptive applications, the proposed approach, at a high level of abstraction, does not depend on any client-side logic. We therefore assume that the core business logic resides on the server side. This is in line with the fact that the classical architecture is still representative of the most widespread type of Web applications.

In this technological context, several conceptual modeling approaches for Web application design [21] build on a strong separation of concerns among *data* and *hypertext* design. Also when modeling context-awareness this separation can be leveraged successfully. Figure 1 graphically summarizes such separation and highlights some issues related to the context data representation and the flow of context data within a context-aware application.

The application's data source includes both the *application data* (i.e., the business objects that characterize the application domain), and a *context model* that offers at any moment an updated representation of the context state. The context model captures all the context-characterizing properties (i.e., attributes

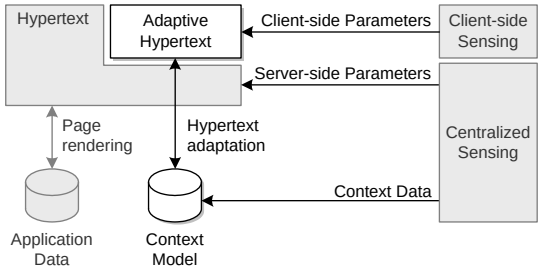


Fig. 1. Context data in context-aware Web applications.

and/or changes in time) that enable the system reactivity. This organization of data requires the following issues to be addressed:

1. *Context model definition and representation* within the application data source. The main context properties needed for supporting adaptivity must be identified and represented as data.
2. *Context model management*, consisting of:
   (a) *Context data acquisition* by means of measures of real-world, physical context attributes, characterizing the usage environment. Measures can be performed at client side (e.g. by means of GPS-equipped devices) or by means of centralized, dedicated infrastructures (e.g. an RFID infrastructure) that communicate context data directly to the application server. The acquired data are then used to update the context state maintained in the context model.
   (b) *Context data monitoring* to detect those variations in context data that trigger adaptivity actions. Any variation may cause an automatic (*context-triggered*) adaptive behavior of the Web application.

At runtime, the dynamic computation of the hypertext interface primarily makes use of application data for populating pages with contents. However, a subset of hypertext pages, the *adaptive hypertext*, is augmented with some *adaptive actions*, whose computation also exploits context data. Adaptive actions can be applied to:

1. *Contents* and *services* delivered by accessed pages, which are customized on the basis of the current context;
2. The *navigation*, by means of automatic navigation actions towards pages of the same application, better suited to the current context conditions;
3. 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. *Presentation properties*, in order to provide more fine-grained adjustments of the application's appearance.

Context monitoring mechanisms are required to identify the variations of some context properties, evaluate them against some conditions, and trigger the required adaptivity actions. Context monitoring can be achieved by periodically refreshing pages, evaluating the context and triggering adaptivity actions at any page request. This is the most frequent solution adopted so far. However, in addition to the (passive) hypertext generation methods, an *active* mechanism should be able to operate autonomously and transparently in the background, independently from the user- or the refresh-based request of Web pages.

In the following sections, we will illustrate how the previous requirements, from context data capturing and representation to the specification and enactment of adaptivity actions, can be captured by a development methodology supplying a conceptual model for data and hypertext design, and a set of transformations for automatic code generation implemented within a CASE tool.

3 Designing the Context Model

According to the previously described separation of concerns, in our approach the *context model* has a counterpart at both data and hypertext level. At data level, context is represented by a set of data entities and relationships that extend the application data schema; we call this extension *context sub-schema*. In our previous work [7], we have investigated adaptivity solutions storing context data at client side (mainly the user profile). The experiment consisted in the generation of client-side guides for the adaptive visit of Web applications. In that experiment, storing the user profile at client side was particularly advantageous, since the generation of the guides was entirely managed by the client. In this paper we instead assume the context state be maintained at server side, because, as better explained in Section 4.2, the enactment of adaptive behaviors is tightly coupled with page computation, which occurs at server side.

Figure 2 illustrates an ER diagram exemplifying a possible context sub-schema; it includes entities, such as `Device`, `Location` and/or `Activity`, that describe particular properties of the context needed by the application for adaptivity purposes.

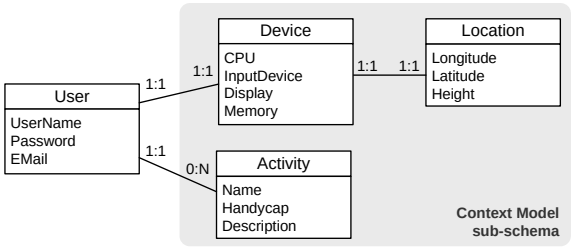


Fig. 2. Example context model as ER diagram.

The context model may vary depending on the application domain and, also, on the adaptivity goals to be fulfilled. In principle, several properties are commonly regarded as *context attributes* (e.g. position, time, or device characteristics) [37]. In practice, there exists no universal context model that applies to all kinds of applications. Therefore, our approach just prescribes to have *user* context entities (e.g., the position or the device of an individual user) associated, directly or indirectly (as in the case of the entity `Location` in Figure 2), to the entity `User`. Starting from the identity of a user, it is possible to navigate the context model and to extract individual context information. *System* context entities (e.g., the current load of a monitored system resource, such as a CPU) that do not represent personalized context data can be accessed independently form the user and therefore do not require any association to any particular user.

Orthogonally, it is also possible to distinguish between *physical* and *logical* context [35]. The former refers to raw sensed data that can be captured both

at client-side (as in the case of GPS-based coordinates) or at server-side (as in the case of radio signals in RFID-based sensing infrastructures). The latter refers to concepts associated to physical data (e.g., core application data, such as the building placed at some GPS-based coordinates), which provide meaningful abstractions with respect to the application domain and the needs of the users interacting with the application. Typically, applications react to changes in the logical context that – in order to be signaled to users – really ask for the adaptation of a viewed page.

4 Modeling Context-Aware Hypertexts

For the design of adaptive hypertext front-ends, we have extended WebML [9], an already established model for the design of data-intensive Web applications. Before concentrating on the extensions introduced for context-aware Web applications, in the following section we briefly recall the basic WebML concepts.

4.1 WebML Overview

WebML is a visual language for specifying the content structure of Web applications and the organization and presentation of contents into one or more hypertexts [9, 11].

WebML design starts with the specification of a *Data Model*, expressing the organization of the application contents by means of a well established data model, such as the Entity-Relationship diagram. Then, the WebML *Hypertext Model* allows describing how contents, previously specified in the data schema, are published into the application hypertext. The overall structure of the hypertext is defined in terms of *site views*, *areas*, *pages* and *content units*. A *site view* is a hypertext, designed to address a specific set of requirements. Several site views can be defined on top of the same data schema, for serving the needs of different user communities, or for arranging the composition of pages to meet the requirements of different access devices like PDAs, smart phones, and similar appliances.

A site view is composed of *areas*, which are the main sections of the hypertext, and comprise recursively other sub-areas or pages. *Pages* are the actual containers of information delivered to the user; they are made of *content units*, which are the elementary pieces of information extracted from the data sources by means of queries, and published within pages. In particular, content units denote alternative ways for displaying one or more entity instances. Unit specification requires the definition of a *source* and a *selector*: the source is the name of the entity from which the unit's content is extracted; the selector is a condition, used for retrieving the actual objects of the source entity that contribute to the unit's content.

Content units and pages are interconnected by *links*, which represent navigation alternatives. Links can connect units in a variety of configurations, yielding

complex navigation structures. Besides representing user navigations, links between units also specify the transportation of parameters to be used by the destination unit in the selector condition for extracting the data instances to be displayed.

Some WebML units also support the specification of content management operations. They allow creating, deleting or modifying an instance of an entity (respectively through the `Create`, `Delete` and `Modify` units), or adding or dropping a relationship between two instances (respectively by means of `Connect` and `Disconnect` units).

Besides having a visual representation, WebML primitives are provided with an XML-based textual representation, which specifies additional detailed properties, not conveniently expressible in the graphic notation. Web application design based on WebML can therefore be represented as visual diagrams, as well as XML documents. The XML representation constitutes the starting point for the automatic generation of the application code to be executed by means of a proper runtime environment. For a more detailed and formal definition of WebML, the reader is referred to [9].

4.2 Modeling Context-Awareness

As illustrated by the WebML hypertext schema in Figure 3, our basic assumption about context-aware hypertexts is that context-awareness is a property to be associated only to *some* pages of an application, not necessarily to the application as a whole. Location-aware applications, for example, adapt "core" contents to the position of a user, but "access pages" (including links to the main application areas) are typically not be affected by the context of use.

We tag adaptive pages with a `C`-label (standing for *context-aware*) for distinguishing them from conventional pages. The label indicates the association of the page with a *Page Context*, i.e., a page-specific set of parameters of physical sensed data and/or of logical context data that are monitored to trigger a page's

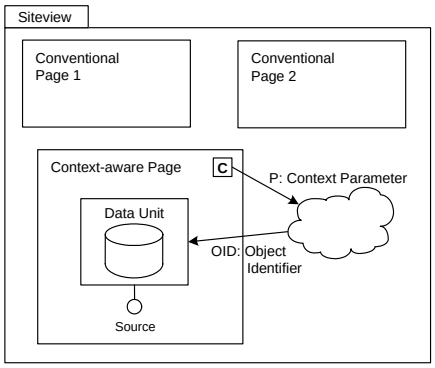


Fig. 3. Coarse hypertext schema highlighting conventional and context-aware pages.

adaptivity features while the page is visited. Page Context parameters correspond to those physical or logical context attributes in the context model whose variation effectively demands for a re-computation of the currently viewed page.

The state of the Page Context is observed by a *Context Monitor*, for detecting significant changes demanding for adaptivity. During the user's visit to a C-page, such changes generate the automatic request of the page, thus the evaluation and/or execution of the adaptivity actions associated to the page. Evaluation occurs prior to the actual page computation, as the actions might serve for customizing the page content or for modifying the navigation flow.

The design of adaptivity actions assumes a central role. As shown in Figure 3, we abstract the adaptivity actions associated to pages by means of so-called *context clouds*. Clouds are external to their pages and host the respective adaptivity actions. While pages act as providers of contents and services, context clouds act as modifiers.

As further shown in Figure 3, in our notation, adaptivity actions clustered in the context cloud are associated to a page by means of a directed arrow, i.e., a link exiting the `C`-label, which represents their automatic triggering that occurs when the Page Context changes. This link ensures the communication between the page logic and the cloud logic, as it allows transporting parameters deriving from page contents and required for computing the actions specified within the cloud. Also, on the other way around, a link from the cloud to the page can transport parameters computed by the adaptivity actions, which might serve for adapting page contents with respect to the updated context.

Analogously to the case of `C`-pages, context clouds may further be associated to entire areas or site views in a WebML hypertext schema, thereby defining common adaptivity actions to be applied to each of the contained `C`-pages. In presence of multiple clouds to be evaluated for a specific `C`-page (i.e., in presence of nested `C`-containers), the evaluation recursively considers each of the clouds, starting from the outermost one and ending with the one associated to the page. This convention allows designers to reduce schema redundancy and to enhance the readability of hypertext schemas.

For the specification of adaptivity actions, some new WebML units have been defined (see Figure 4 and Figure 5 for their visual notation and description). Together with standard WebML content and operation units, they allow us to address the two main requirements posed to context-aware Web applications, as described in Section 2: *context model management* and *hypertext adaptivity*.

Context Model Management. To assure that application adaptation occurs always on top of an up-to-date picture of the current context, context model management operations are the first actions executed every time a context-triggered access to a `C`-page occurs. Issues for context model management, already addressed in Section 2, are modeled as follows:

– *Acquisition of client-side-generated* context data is modeled through the `Get URL Parameter` unit (see Figure 4). In particular, the unit represents the access to fresh context data sensed at the client side and transmitted by

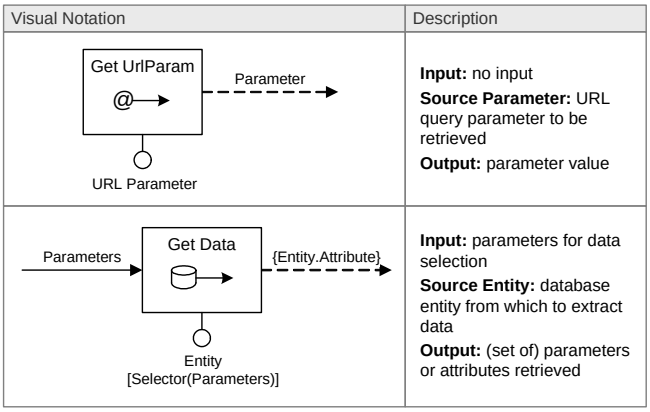


Fig. 4. New defined units for context model management.

means of device- or client-side-generated parameters, appended to the page request string.

- *Access to context data in the context model* is modeled through the new `Get Data` unit (see Figure 4). Usually, page computation implies retrieving data to be published within pages. The execution of adaptivity actions may in addition require the retrieval and evaluation of context data stored in the context model, with the only purpose of evaluating conditions that may activate the adaptivity actions, without requiring any visualization. The `Get Data` unit allows conditions to access such data. This new construct also supports the access to fresh context data in all those situations where a centralized sensing infrastructure is adopted.
- *Context model updating* operations, needed after the acquisition of fresh context data, can be specified by means of WebML operation units (i.e., `Modify` or `Connect` units).
- *Context model monitoring* is achieved by extending the page logic through a periodic checking mechanism for querying the context model and identifying any variation of the page context that requires the execution of adaptivity actions.

Hypertext Adaptivity. Some WebML constructs (e.g., operation units or links) allow designers to visually define adaptivity actions within context clouds. The computation of context clouds is triggered by the Context Monitor, based on the evaluation of the current Page Context. The execution of the adaptivity actions within the clouds may further depend on the evaluation of some *conditions*, are modeled using WebML constructs, such as `If` and `Switch` units. If the conditions are satisfied, several actions can be performed:

- Adaptation of *page contents*. Parameters produced during context data acquisition and through condition evaluation can be used for page computa-

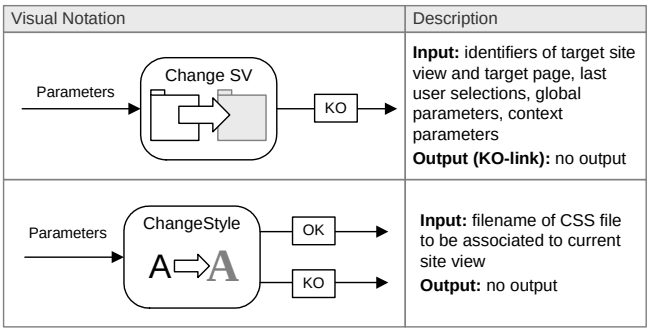


Fig. 5. New defined units for hypertext adaptation.

tion. The result is a page where contents are *filtered* with respect to the current context.

- Adaptation of *navigation*. In some cases, the effect of condition evaluation within the context cloud can be an automatic, *context-triggered* navigation, causing the redirection to a different page. The specification of context-triggered navigations just requires connecting the last link of the context cloud action chain to an arbitrary destination page of the hypertext, for redirecting the user to that page. Therefore, links exiting the context cloud and directed to other pages than the cloud's source represent *automatic navigation actions*.
- Adaptation of the hypertext *structure*. This allows designers to address coarse-grained adaptivity requirements, as required, for example, in the case of changes of the user's device, role and/or activity within a multi-channel, mobile environment. The new `Change Site View` unit in Figure 5 allows the application to switch between site views tailored to different requirements. The site views involved in the switching process must be defined (and detailed) in advance and are not computed during runtime.
- Adaptation of *presentation properties*. More fine-grained adjustments of the application's appearance can be achieved by means of the so-called `Change Style` unit (see Figure 5), which allows changing at runtime the application's CSS (*Cascading Style Sheet*) file.

5 Implementing Active Context-Awareness

When designing context-aware Web applications, different policies can be adopted regarding when adaptivity should be performed. Indeed, not always the request of a C-page necessarily implies the evaluation of its context cloud. For example, in order to give high priority to a user's selection, independently from the current context state, a *deferred* adaptivity policy can be adopted. According to this policy, when the user requests a C-page, the page is generated by considering the only parameters deriving from the selections performed by the user

in a previous page, and the context model is not queried, nor are the adaptivity actions triggered. Only afterwards (after a pre-defined time interval, or on user's request) the requested page becomes subject to adaptivity and updates its contents according to the current context state.

The opposite design choice, denoted as *immediate* policy, grants to context a higher priority. In this case, every time the page is accessed, the context model is queried and the context cloud is processed prior to the page contents. Such policy could be suggested, for example, to handle exceptional situations where a timely reaction to context changes could be more important than following a user's navigation.

Consider for example a tourist guide that shows contents about the attractions located close to the user. At a given point, the user might want to get information about one monument located in a different city area, not related to her/his current position; this preference is typically expressed by selecting a link to that monument from a list of city attractions. In a deferred policy, the requested page shows the monument information as requested by the user, without taking into account the user's current location. Only after expiration of the refresh interval, the page becomes again subject to adaptivity and the contents are adapted to the user's location. In an immediate policy, context is granted higher priority with respect to the user and, thus, the user's request for the monument would be overwritten by the context and the application would show once more the monument associated to the user's current location. Users, of course, can disable and enable adaptivity at will.

We thus distinguish two types of page accesses, *user-triggered* and *context-triggered*, depending on the actor (the user or the context) that has priority according to the adopted policy. A sensible design of adaptive features and a suitable selection of adaptivity policies (for each page) should allow designers to minimize application behaviors that could be perceived as invasive or annoying by users.

According to previous observations, adaptivity actions associated to a given page must be evaluated and executed only in case of context-triggered requests. To distinguish context-triggered page requests from user-generated ones, we append to automatic (i.e., context-triggered) requests a special parameter (named `automatic`), which is not present on user-navigable links. Page requests are thus user-generated if they lack the `automatic` parameter and automatically generated in case they carry the parameter. Accordingly, adaptivity actions are evaluated only for those requests carrying the `automatic` parameter. In order to put computation under the user's control, dedicated links can be arbitrarily set on each page, allowing users to explicitly ask for adaptation (e.g. a page refresh with appended `automatic` parameter).

In our previous work on context-aware Web applications [6], the implementation of the adaptive behavior was based on periodic refreshes of adaptive pages. This resulted in a continuous reloading of viewed pages and, consequently, in a poor usability of the overall adaptive system. To overcome this shortage and to foster *active* adaptivity, we introduced the *Context Monitor* (CM) module,

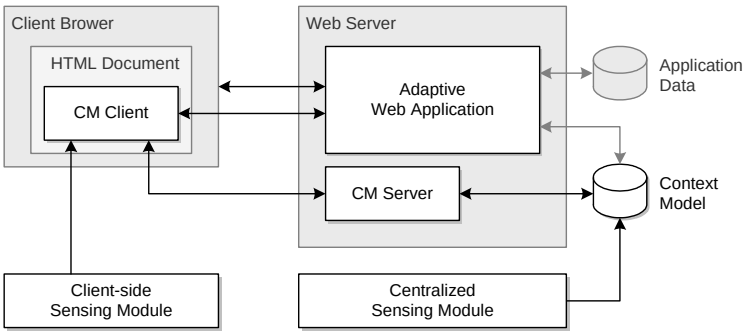


Fig. 6. Functional architecture for active context-awareness.

with the main goal of triggering the evaluation of adaptivity actions (and thus the refresh of the page) only when really required, i.e., only in case of significant context changes, even in absence of user interactions.

Figure 6 shows the resulting functional architecture. The CM consists of two separate modules, one on the client side and one on the server side. The CM Client module is a piece of business logic embedded into the page's HTML code and executed at the client side (e.g. a JavaScript function, a Java applet, or a Flash object), while the CM Server module works in parallel to the Web application on the same Web server. The CM Client periodically polls the server[2] to monitor the context and to decide whether possible context variations demand for the adaptation of the currently viewed page. When the application makes use of a client-side sensing, the CM Client is also in charge of communicating fresh context data sensed at the client side (by querying the client-side sensing module).

In order to take a decision about triggering or not adaptivity actions, the CM Client is assisted by the CM Server, which has full access to the context model of the application maintained at the server side. In response to the polling executed by the CM Client, the CM Server queries the context model and computes a numeric digest over the Page Context parameters of the currently viewed page. The CM Server needs thus to be configured with suitable queries for each context-aware page of the application, as further exemplified in Section 7.2.

The context digest is the basis for the decisions of the CM Client, since it identifies whether variations in the physical context (represented by the fresh context data, such as longitude and latitude) also correspond to variations in the logical context (represented by the Page Context parameters, such as roads and buildings). Depending on the application design, adaptivity must be trig-

[2] The polling of context data, which is imposed by the restrictions of the HTTP protocol, is performed in the background. A "real active" notification initiated by the server and sent to the client would require the client to constantly keep open a specific TCP port, which – due to security risks – in our view is not realistic. The proposed solution is the best one in presence of a clear conflict.

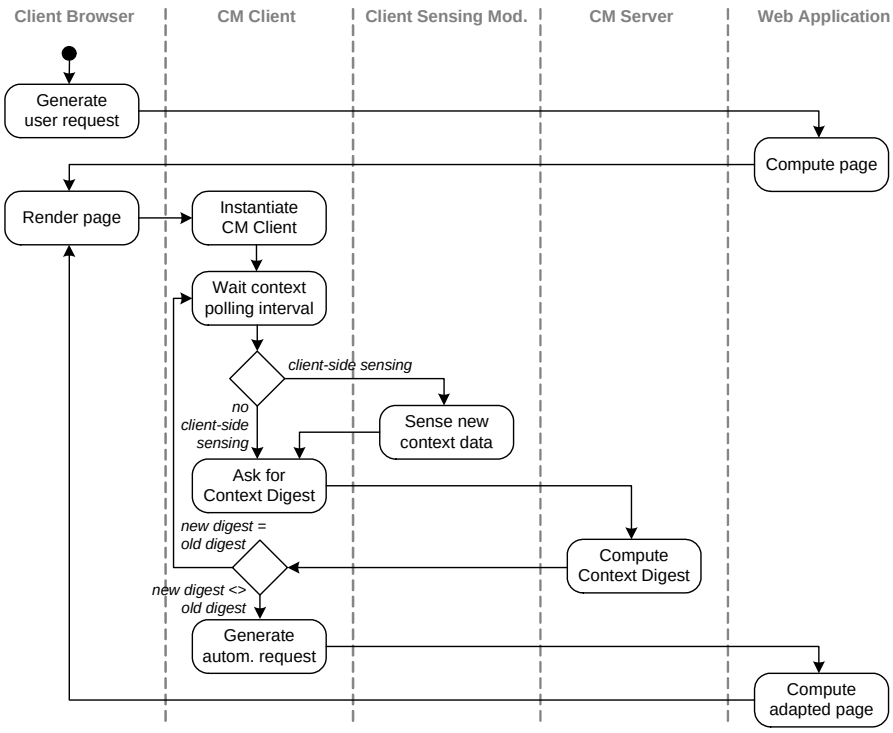


Fig. 7. Active context-awareness (with client-side context sensing): communicating context data and triggering adaptivity.

gered only when logical context variations occur. Logical context parameters supply data that actually influence the page computation. A page update is thus required only if their value has changed.

The context digest is computed over the Page Context parameters, expressed by means of parametric queries over the context model, where the parameters correspond to raw sensed data. The queries thus return the values of the Page Context parameters, possibly in function of the sensed physical context. If such values change from one polling to the other, it means that at least one of the Page Context parameters has changed; the CM Client thus asks the Web application for an adapted version of the page. If the digest does not change, the CM Client proceeds with evaluating the next context digest, and with sending fresh context data in case a client-side sensing is adopted.

Figure 7 details this active behavior cycle by means of an activity diagram and shows how the single modules cooperate in order to determine whether adaptivity is required or not. The diagram has one start node (i.e., `Generate user request`), which corresponds to the user's navigation to a C-page, and no end node, since the cycle in the lower part of the diagram is only interrupted by

another user navigation that may lead the user to another C-page (which would correspond to the start node of the diagram) or to a conventional page.

Note that the client-side sensing module and the sending of fresh context data only apply to applications that sense context at the client side; if only a centralized or server-side sensing mechanism is adopted, the step `Sense new context data` and the possible communication of fresh context data are omitted. For example, a centralized, RFID-based sensing infrastructure does not require any transportation of context parameters from the client to the server. In this case, we assume that dedicated software modules, which are part of the sensing infrastructure, interact with the application and directly update the context model, thus making fresh sensed data available in the data source.

The mechanisms previously described assume that connectivity is available in order for the CM client to communicate with the CM server. In case of intermittent connectivity, which is a very frequent situation in mobile environments, the CM client keeps working by periodically polling the CM Server, despite the absence of connectivity. The CM Client is programmed to manage possible lacks of connectivity; it therefore does not generate errors, with the only side effect that adaptivity is suspended until the connectivity is restored.

5.1 Context Monitor Implementation

The CM has been implemented as client-server module, completely independent from the implementation of the Web application. Despite its integration into the WebML runtime environment as described in this paper, its function is general in nature and, assuming that the access to the context model is granted, only demands for a mechanism to trigger adaptivity actions.

In our current implementation, the CM Client is a Macromedia Flash object[3]. Its configuration is performed directly within the HTML code sent to the client browser, and mainly consists in specifying the context parameters to be sensed at the client side, as well as a suitable polling interval.

The CM Server, on the other hand, is implemented as a Java servlet communicating with the CM Client via the *Flash Remoting Gateway*[4]. The CM Server generates the digests representing a fingerprint of the current context state corresponding to the currently viewed page. The digest computation is based on an XML configuration file, defined for each `C`-page, that contains:

1. The set of Page Context parameters needed for computing the context digest.

[3] Other client-side solutions have been investigated as well: *JavaScript* does not allow reading from the local harddisk for accessing client-side sensed context data; *Java applets* do provide access to local resources, but loading the Java Virtual Machine noticeably delays the execution of applets, especially on small devices such as PDAs.

[4] Flash Remoting is an essential part of Macromedia's approach towards Rich Internet Applications. Flash Remoting for J2EE consists in a single servlet acting as gateway towards the application server's resources, and serves the purpose of de-serializing the proprietary Macromedia AMF (*Active Message Format*).

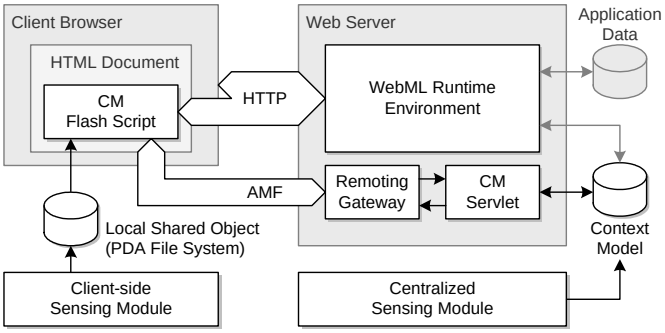


Fig. 8. Context Monitor Implementation.

2. For each Page Context parameter, the query for extracting the respective value from the context model. The query conditions consist in comparing fresh context data with the previous context state. Therefore, they may also use client-side context parameters - if any.

Figure 8 graphically depicts the described implementation of the Context Monitor. The adaptive and non-adaptive hypertext pages, as well as the adaptivity actions, as specified in the previous section, are hosted and executed by the *WebML Runtime Environment* [15, 41], which is part of the deployment architecture described in the following section.

6 Automatic Code Generation

To validate our proposal, we conducted some code generation and runtime experiments that confirmed us the soundness of the methodology. In our approach, code generation is based on *WebRatio* [41], a CASE tool for WebML that supports the visual design of the application schema and the automatic code generation from WebML schemas [41]. Code generation is based on parametric code components corresponding to WebML units and on a proprietary, extensible runtime engine for the Jakarta Struts framework [15]. Parametric components are configured at runtime using XML descriptors that contain SQL queries and parameters for retrieving contents from the application data source.

The implementation of the extension introduced in this paper exploits WebRatio's native extension mechanism that allows one to add new features by means of so-called *custom units*, a mechanism that has already demonstrated its power when extending the CASE tool to support other functions, such as Web services [3]. The so achieved extension fully reflects the proposed (visual) design method, and paves the road for the automatic generation and deployment of active, context-aware Web applications. In particular, the extension of the WebRatio tool occured along the following three dimensions:

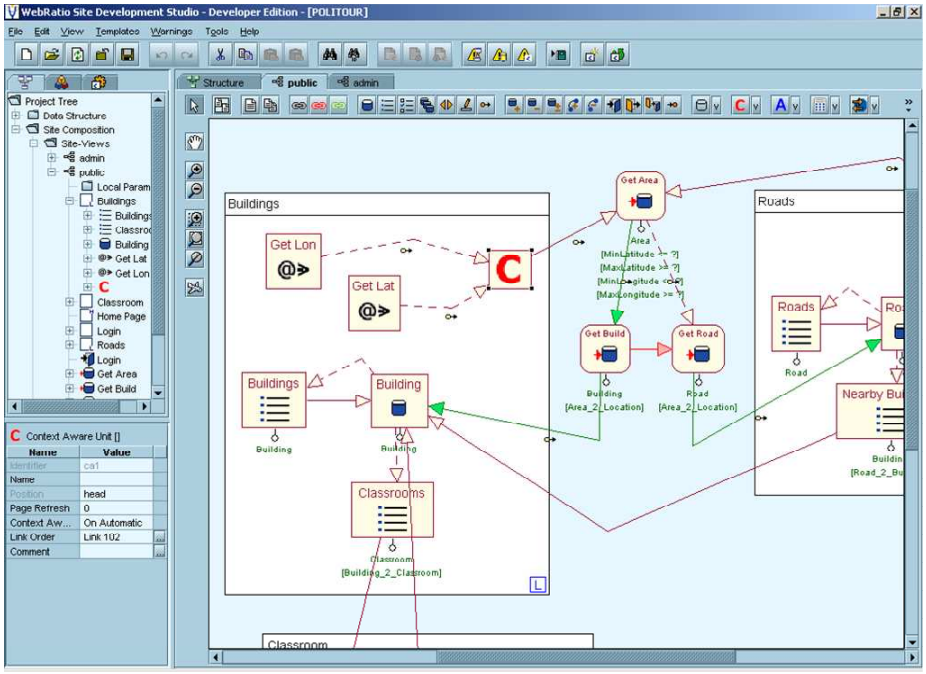


Fig. 9. The WebRatio modeling tool, extended with the new adaptivity-supporting units.

- The extension of the WebRatio visual environment by introducing the new visual primitives. The screenshot in Figure 9, for example, refers to the extended WebRatio visual environment and shows a WebML model fragment of the context-aware Web application illustrated in the following case study.
- The extension of the WebRatio code generator to automatically generate descriptors for the new units.
- The extension of the WebRatio runtime engine to properly interpret generated unit descriptors during application execution.

Active context-awareness has been achieved by revising the logic of the page computation, yielding a further new unit (called `Context` unit) in addition to the units already discussed in Section 4.2, to be used in place of the `C`-label associated to context-aware pages. This unit triggers the context cloud logic at each context-triggered page request, and manages the parameter passing between page and context cloud. Furthermore, the `Context` unit supports the configuration of the CM Client, required for the automatic inclusion of such a piece of business logic into adaptive pages.

The current extension of the WebRatio CASE tool provides support for the automatic generation of the (adaptive) hypertext and the automatic, page-specific configuration of the CM Client module. The configuration of the CM Server module consists in the declaration of the Page Context parameters that

are used to compute the context digest and in the specification of the respective DB queries that associate a value to each parameter. At the current state of the work, this configuration still needs to be hand-coded by designers, but we envision a visually assisted specification (e.g. by means of a proper wizard) of Page Context parameter queries. A similar approach is for example already supported by WebRatio for the specification of derived relationships between data entities. Finally, since code generation starts from WebML schemas, only WebML-related concepts can be automatically coded. The configuration of and the interaction with application-specific context sensors is thus out of the scope and always requires an applicaton-specific treatment. In fact, each application may require different sensors on either client or server side, with different interaction modalities and software support (if any). Our solution provides a simple interface to exchange context data (at server side, the context model in the application's data source; at client side, a shared file that can be read by the CM Client), but the proper use of these interfaces must be realized for each sensor the application requires.

7 Case Study

The research described in this paper was conducted in the context of the Italian research project MAIS (Multichannel Adaptive Information Systems [5]), officially concluded in June 2006 with a publicly accessible demo and presentation day held at the Politecnico di Milano, Italy. To demonstrate the viability of our work and the extension of the WebRatio CASE tool, we implemented a context-aware proof-of-concept application called *PoliTour*, supplying location-aware information about roads, buildings and classrooms in the university campus. During the workshop day, the demo was also complemented with a demonstration of the overall methodology for the development of context-aware Web applications.

The event was held in one of the arcades of the Politecnico campus, which for the event has been covered with a WiFi connection. Using this WiFi connection, the PoliTour application could be accessed through a PDA, equipped with a GPS receiver for (outdoor) location sensing. In PoliTour, sensed context data are thus geographical longitude and latitude for user positioning, but also the signal strength (RSSI) of the available WiFi connection to alert the user of low connectivity areas; both position and signal strength are sensed at the client side. Accordingly, as the user moves around the campus, the application publishes location-aware details about the campus infrastructure and alerts the user in case s/he is about to leave the WiFi-covered area. The former adaptivity implies automatic changes of visualized contents and automatic navigation actions, while the latter is achieved by appropriately changing the CSS style sheet associated to the application (e.g., changing the application's background color).

Figure 10 illustrates the data schema of the PoliTour application, where the entity `User` represents the user model and the entities `Location` and `Connectivity`, associated to `User`, constitute the context model. In order to translate sensed,

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

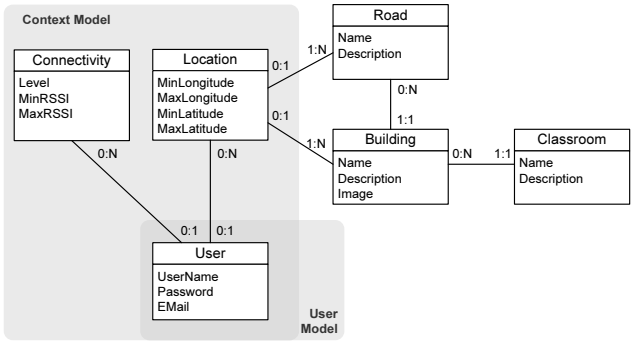


Fig. 10. PoliTour data model.

physical position data into meaningful location information that can be used to determine the adaptive behavior of the application, the campus can be divided into a set of contiguous, rectangular areas, and roads and buildings can be mapped onto those areas. Longitude and latitude allow thus the application to identify the specific rectangle, which on the other hand allows the application to identify a building or road. Therefore, four attributes characterize the entity `Location`: min and max longitude, min and max latitude. The entity `Connectivity` represents the translation of the WiFi connection quality from the sensed, continuous RSSI signal values into discrete, qualitative levels (i.e., "Low" and "High"). The entity is described by three attributes `Level`, `minRSSI` and `maxRSSI`.

The remaining entities `Road`, `Building` and `Classroom` refer to the application data, as they are not strictly related to the context-aware behavior of the application and can be navigated by users as core contents of the application. Note, however, that the entities `Road` and `Building` are also part of the logical context, as they store data that are location-dependent.

While the entity `Connectivity` contains the necessary logic to translate physical RSSI data into a logical RSSI level, the entity `Location` only translates the physical position into a first level of logical context, that is geographical areas inside the university campus. But as the application reacts to changes of buildings or roads and not just areas, the retrieved areas must be further translated into buildings or roads by navigating the relationships from the context model to the application data. Hence, since adaptivity may be also based on application data, not just immediately available context data, the context model of the application needs to be kept on the server side where the application data resides.

Figure 11 shows a simplified WebML hypertext schema of the PoliTour application; for presentation purposes, we only show three pages of the overall application. Let's first concentrate on the hypertext elements without adaptive behavior: Page `Buildings` in the lower left part of the figure publishes a list of buildings (`BuildingsIndex`). It also shows details of a selected building

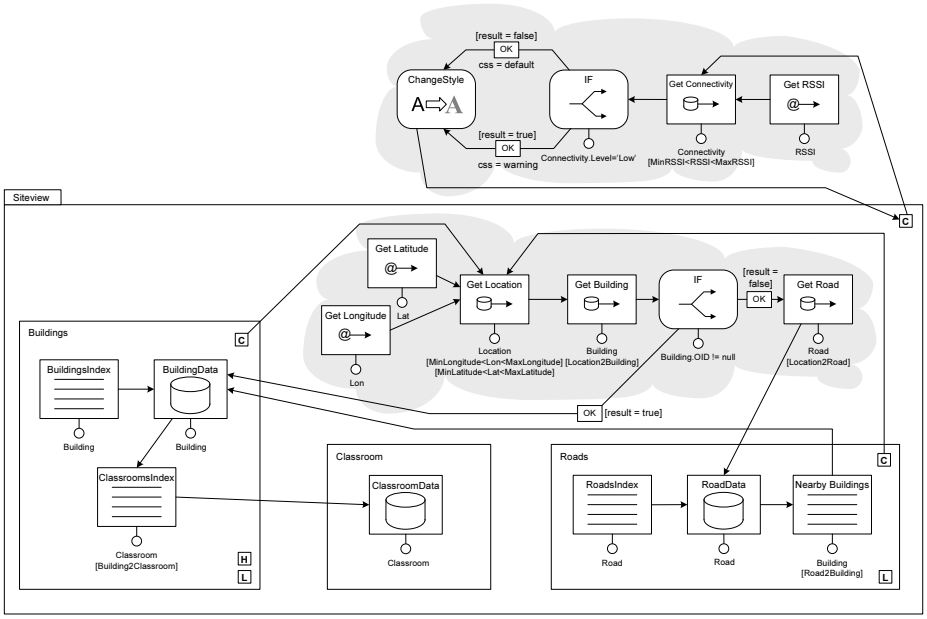
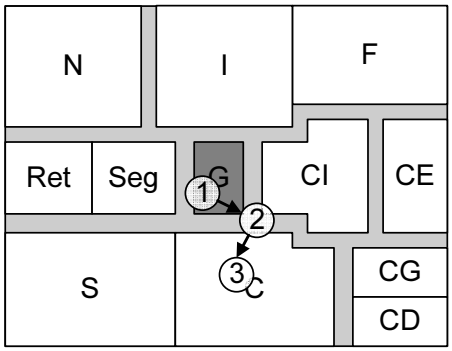


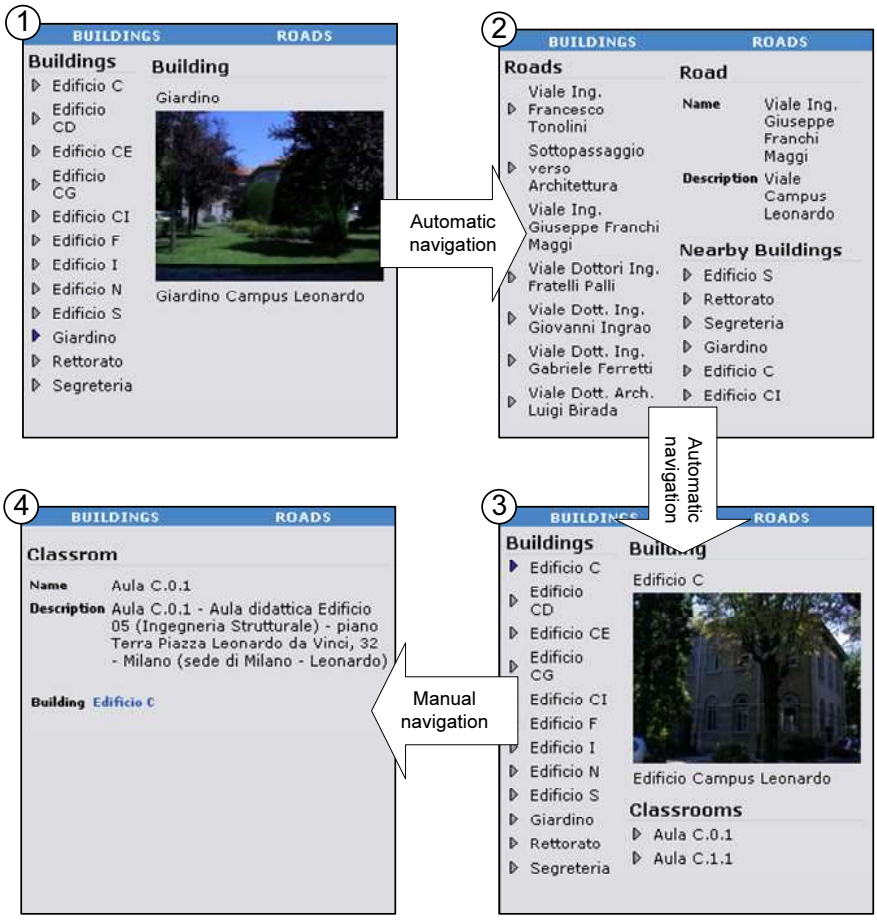
Fig. 11. PoliTour hypertext model.

(`BuildingData`) chosen from the list, together with the building's classrooms (`ClassroomsIndex`). The selection of a classroom leads the user to a new page (`Classroom`, in the lower center part of the figure), which shows details about the selected classroom (`ClassroomData`). Similarly to page `Buildings`, page `Roads` (in the lower right part of the figure) shows data about roads (`RoadsIndex` and `RoadData`) and their nearby buildings (`NearbyBuildings`). A user can for example select a specific road from the list `RoadsIndex` to view its details in the `RoadData` unit and its nearby buildings in the `NearbyBuildings` unit. The pages `Buildings` and `Roads` are so-called *landmark* pages, which can be accessed through links grouped in a menu displayed in every application page. The page `Classroom` instead can only be accessed through a link in page `Buildings`, which is also the home page of the whole site view.

Let's now discuss how context-awareness is modeled in the hypertext schema: The pages `Buildings` and `Roads` are `C`-pages; starting from the respective C-labels, the pages are connected to the context cloud outside the pages (but still inside the site view) that represents their adaptivity logic. In particular, the units `Get Longitude` and `Get Latitude` access the physical location data sensed through the client-side GPS module, the `Get Location` unit extracts the corresponding logical area from the context model, and the `Get Building` unit, finally, retrieves the building contained in the identified area. If no building is associated with the current location, which is checked by means of the `If` unit, the `Get Road` unit retrieves the road associated with the current position, and



(a) Main campus map of Politecnico di Milano.



(b) Screenshots of a typical use of the application.

Fig. 12. The running PoliTour application.

forwards the user to the `Roads` page. If instead a building is retrieved, page `Buildings` displays the respective details. If from the `Buildings` page a user navigates to the `Classroom` page, the context-aware behavior of the application is interrupted, as the `Classroom` page is not defined as context-aware. As soon as the user turns back to the `Buildings` or `Roads` page, the adaptive behavior is again enabled.

Figure 12 exemplifies a possible interaction with the PoliTour application. Let's assume the Politecnico campus is organized as represented in Figure 12(a), and that the user wants to move from location 1 to location 3, as highlighted on the map. Figure 12(b) shows the respective screenshots. The user starts from the central garden in the campus, moves to a nearby road and, finally, enters building *C*. The application automatically adapts the published contents accordingly. Once in the building, the user selects one of the classrooms of the building (see screenshot 4), thus s/he accesses to the non-adaptive page `Classroom`. Turning back to one of the two context-aware pages `Buildings` or `Roads` would again enable the automatic adaptation of contents.

To visually alert users of low connectivity, we specify a second adaptivity cloud that takes into account the sensed RSSI level and, in case of low connectivity, changes the CSS style sheet of the application. In order to specify the cloud only once for all C-pages of the site view, we associate it to the site view instead of to each single page. The unit `Get RSSI` accesses the WiFi signal strength registered on the PDA, and the `Get Connectivity` unit translates the sensed value into a corresponding connectivity level. If the retrieved level is "Low", the `Change Style` unit sets the current CSS file of the site view to "warning", which means having a red page background. Otherwise, the "default" style sheet with a gray page background is adopted. Since this cloud is associated to the site view, it is evaluated before the evaluation of any possible context cloud associated to C-pages.

7.1 Configuration of the CM Client

The configuration of the CM Client for the PoliTour application is achieved by means of the following (HTML) code lines to be added when rendering the page:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
        id="CMClient">
<param name="movie" value="CMClient.swf">
<param name="flashvars"
       value="gatewayURL=http://dblambs.polimi.elet.polimi.it/
       demogpspoli/gateway&refresh=5&userOID=1
       &currentURL=
       http://dblambs.polimi.elet.polimi.it/demogpspoli/page4.do
       &contextParams=longitude;latitude;RSSI">
</object>
```

As described in Section 5.1, due to the client-side sensing, client configuration requires to specify: the context parameters (`contextParams`) to be sensed

at the client side, i.e., `longitude`, `latitude` and `RSSI`, and the polling interval (`refresh`), i.e., 5 seconds. These parameters are automatically configured at design time through the WebRatio visual enviroment. The CM Client configuration requires also other parameters to work properly: `gatewayURL` represents the URL of the Flash Remoting Gateway, `userOID` represents the user identifier, and `currentURL` represents the currently requested page. These parameters are automatically configured at run time by the execution environment.

7.2 Configuration of the CM Server

The following XML code presents a fragment of the CM Server configuration file concerning page `Buildings`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ConfigElements>
 <CAPage id="Buildings">
  <PageContextParams number="2">
   <param>
    <name>buildingOID</name>
    <query>
     select BUILDING.OID
     from   BUILDING, LOCATION
     where  LOCATION.toBUILD=BUILDING.OID
     and    LOCATION.MINLONGITUDE<?longitude?
     and    LOCATION.MAXLONGITUDE>?longitude?
     and    LOCATION.MINLATITUDE<?latitude?
     and    LOCATION.MAXLATITUDE>?latitude?
    </query>
   </param>
   <param>
    <name>connLevel</name>
    <query>
     select CONNECTIVITY.LEVEL
     from   CONNECTIVITY
     where  CONNECTIVITY.MAXRSSI>?RSSI?
     and    CONNECTIVITY.MINRSSI<?RSSI?
    </query>
   </param>
  </PageContextParams>
 </CAPage>
 <CAPage id="Roads">
   ...
 </CAPage>
</ConfigElements>
```

In more detail, the code fragment contains the following configurations:

– The context digest is computed over the `buildingOID` and `connLevel` values that represent logical context values (as defined in Section 3) at the right level of abstraction to determine the adaptive behavior of the application.

– The value of `buildingOID` is extracted through a parametric query that uses the `longitude` and `latitude` values provided by the CM Client[6] and retrieves the building associated to the current user location. In case the query does not return any result, the value associated to `buildingOID` is *null*.
– The value of `connLevel` is extracted using the `RSSI` value provided by the CM Client. In case the query does not return any result, the value associated to `connLevel` is *null*.

A similar configuration applies also to page `Roads`, but is omitted in the above code fragment for the sake of brevity.

7.3 Implementation and Deployment

The described application has been modeled according to the outlined approach. Hypertext and CM Client configuration have been automatically generated with the extended WebML code generator and deployed on top of a J2EE platform. The configuration of the CM Server and the interaction with the sensing devices have been coded manually. Its use is possible through PDA devices with wireless Internet access, using *Pocket Internet Explorer* with *Flash plugin*. The communication between the GPS module and the CM Client is implemented using the *Chaeron GPS Library* [12]; the WiFi received signal strength indicator (RSSI) is acquired in the PDA using *Place Lab* [32]. A demo of the PoliTour application is available at `http://dblambs.elet.polimi.it/politour/`.

8 Related Work

Context-awareness in general, until recently, has been mainly studied in the fields of ubiquitous, wearable or mobile computing. A significant number of dedicated applicative solutions have been successfully developed [40, 30], and context abstraction efforts have produced proper platforms or frameworks for rapid prototyping and implementing context-aware software solutions [33]. Within the domain of the Web, so-called *adaptive hypermedia* systems [4] use a user's preferences, knowledge and goals throughout an interaction to adapt the hypertext to the needs of that user. Recent research efforts also address the special needs of portable devices and mobile Web applications.

The *AHA!* system proposed by De Bra et al. [16] represents a user modeling and adaptation tool originally developed in the e-learning domain. According to a continuously updated user model, it allows customizing hypertext links (adaptive navigation) and contents (adaptive presentation). Adaptivity only occurs in response to user-generated page requests.

Belotti et al. [2] address the problem of fast and easily developing context-aware (Web) applications along a technological, database-driven approach, based

[6] Variables expressed as `?name?` refer to client-side sensed context data that the CM Client provides in input to the CM Server at each request for a new context digest.

on extended functionalities specifically tailored to Web publishing. The authors propose the use of a universal context engine in combination with a suitable content management system [23]. In [2] they describe their resulting general context-aware content management system, which enables developers to seamlessly adapt content, view, structure and presentation of Web applications to runtime context properties. Context affects the actual Web application indirectly by altering the state of the database and is not able to trigger autonomously application functionalities.

At a more conceptual level, some well known model-driven methodologies, such as *HDM* [22], *OOHDM* [36], *Hera* [39], and *UWE* [29], aid developers in the design of Web information systems. Despite the growing number of specific applications, only few attempts exists that aim at modeling also active, context-aware behaviors at a conceptual level as described in this paper.

In [20] the authors show how the model-driven *Hera* design methodology allows designers to specify the conditional inclusion or exclusion of page fragments at content, navigation and presentation level. Adaptation is achieved by means of so-called appearance conditions attached to design artifacts and based on user profile and device capability information. Each of the Hera models (conceptual model, application model, presentation model) enables access only to those profile attributes that are meaningful in the context of the particular model. Adaptivity is thus mainly based on user and device data, and customizations according to a broader interpretation of *context* are not addressed.

Fiala et al. [18], on top of the model-based framework of the *Hera* project, propose a component-based XML document format for the implementation and deployment of component-based, adaptive Web presentations (*AMACONT* project). Adaptation depends on user profile data and device characteristics and mainly concerns layout and presentation properties of Web pages. The implementation of AMACONT-based applications is supported by an automatic code generation mechanism for adaptive documents and multiple communication channels, starting from AMACONT components and Hera schemas.

Similarly to Hera, Jin et al. [26] propose an ontology-based, model-driven approach for declarative Web site design: OntoWebber. The authors stress the importance of data integration in the context of Web portals by means of semistructured data formats and equip OntoWebber with a suitable integration layer. Web site design involves the design of several different models: content model, navigation model and presentation model can be customized according to a personalization model. Analogously to our approach, the authors distinguish between fine-grained and coarse-grained adaptation/personalization; the former is achieved by incrementally rewriting a user's site view structure, the latter is achieved by switching between site views. Personalization, however, is based only on three user properties, i.e., capacity, interest, and request, while the approach described in this paper proposes the use of any kind of context data for supporting adaptivity.

Baumeister et al. [1] explore *Aspect-Oriented Programming* [19] techniques for modeling adaptivity in the context of the UML-based Web Engineering

method (UWE [29]). The authors concentrate mainly on aspects for adaptive link hiding, adaptive link annotation and adaptive link generation. This kind of adaptation is achieved in UWE by adding annotations, handled as independent model aspects (the authors distinguish *model time* and *runtime* aspects), to navigation links of the *navigation model*. Content adaptation or presentation adaptation are not tackled yet, but they are under investigation by the authors. The main contribution of the work can be identified in the strong separation of *navigation model* and *adaptation model*, achieved by interpreting adaptation as cross-cutting aspect with respect to application modeling. This aspect is in line with our approach, where context cloud modeling is orthogonal to page modeling.

Schewe et al. [34] describe an algebraic approach to personalization or adaptation in Web information systems by extending SiteLang, a process algebra developed by the authors to express so-called application "stories". User preferences are specified by means of proper pre- and post-conditions that act as filters over a Web information system's story space and that tailor the algebraic expression of a story space to an individual user. Unlike the previous conceptual approaches, this idea leverages formal reasoning about Web information systems, which is based on algebraic expressions that make the implementation less intuitive (compared to model-driven approaches).

Although the previous works aim at providing advanced adaptivity support for Web applications, differently from our approach none of them grants context the status of proper *actor* on top of the actual application. Partly, this can be ascribed to the lack of suitable technologies for client-side adaptivity and/or active communication protocols. The growing interest in so-called *Rich Internet Applications* (RIAs), deployed on top of desktop-like, client-side execution platforms, underlines the need for more powerful Web technologies. In this context, above all the Macromedia MX suite [31] and OpenLaszlo [27] are paving the road for more flexible adaptivity. However, we in general observe a lack of conceptual primitives comparable to the notion of Page Context and Context Monitor. We believe that such or similar abstractions are instead essential to widen the applicability of Web technologies also to non-traditional field, such as active adaptivity.

9 Conclusions and Future Work

In this paper we have considered a relevant aspect of modern Web applications, i.e. adaptivity to context, and we have shown how such issue requires increasing the expressive power of Web application models so as to incorporate context-triggered changes in the page generation logic. The proposed approach, based on WebML, enables the automatic generation of most application components (in the two layers of the data source and of the hypertext front-end), which allow us to achieve some active context-awareness features. However, the proposed design primitives are general in nature, and context modeling, adaptation changes, parameter passing, and the Context Monitor can be manually encoded

by Web programmers. Even in this case, the proposed modeling method still constitutes a valuable support. It clarifies issues behind context-awareness for Web applications and makes application development systematic.

The modeling primitives that we have proposed allow us to cope with some relevant requirements that are generally valid in any class of context-aware applications, namely with context acquisition and monitoring, and with the execution of adaptivity actions along some well-acknowledged dimensions of Web applications. However, new primitives could be needed for modeling new requirements possibly emerging in specific application domains. In this case, the WebML method and the accompanying CASE tool are still able to support an easy extension of the modeling language through the definition of plug-in units [9].

Despite the value of our work in providing conceptual abstractions, methodological tools and techniques for managing context-awareness in Web applications, we believe that some efforts must still be devoted to make it more exhaustive. In particular, some limits of our solution refer to the coverage of the possible adaptive behaviors that can occur in a context-aware Web application. In [13] we have investigated the potential of (textual) Event-Condition-Action rules in combination with the (visual) solution described in this paper. This implies interfacing the application runtime environment with a decoupled rule engine for rule evaluation and action enactment [5, 13]. Such an integration leads to the following advantages:

- The availability of a detached rule engine widens the set of events to react to, also comprising events that are independent from user actions.
- The resulting architecture enhances separation of concerns, and supports flexibility and evolvability: Thanks to the availability of a detached rule engine, the modification or the addition of rules can be managed even after application deployment by properly setting the external rule engine, without requiring changes to the application design and the generation of a new application version.
- The decoupled rule engine also facilitates the management of rule priorities and conflicts, without burdening the Web application computation.

We are currently implementing the rule engine for covering sparse adaptivity [13]. We are also studying adaptivity modes that are either time-dependent (i.e., whose polling intervals can be modeled according to given policies) or real-time (i.e., whose triggering is immediate thanks to context events). Further efforts will investigate the potential of post-processing mechanisms for fine-grained adaptation of presentation properties, i.e. adaptive link hiding. Nevertheless, such new extensions do not compromise the computation logic for C-pages and the mechanisms for context monitoring that we have described in this paper, which therefore will keep their validity even in the extended adaptivity framework.

The solutions described in this paper have been extensively tested in the context of the MAIS project (`http://www.mais-project.it`), by extending the run time component of WebRatio (definition of new units) and accompanying

it with the Context Monitor module. This has allowed us to prove that the solution is feasible and meets an important customer demand. As such, it will be integrated in a future release of the WebRatio environment.

Acknowledgment
We acknowledge the contribution of Marco Valenti to the Context Monitor and the prototype implementation.
This work has been supported by the Italian FIRB Project MAIS (Multi-channel Adaptive Information Systems).

References

1. H. Baumeister, A. Knapp, N. Koch, and G. Zang. Modeling Adaptivity with Aspects. In *Proceedings of the International Conference on Web Engineering - ICWE 2005*, Sydney, Australia., *LNCS* 3579, pages 406–416. Springer-Verlag Berlin Heidelberg, July 2005.
2. R. Belotti, C. Decurtins, M. Grossniklaus, M. C. Norrie, and A. Palinginis. Interplay of Content and Context. In *Proceedings of the International Conference on Web Engineering - ICWE'04*, pages 187–200, 2004.
3. M. Brambilla, S. Ceri, P. Fraternali, R. Acerbis, and A. Bongio. Model-driven Design of Service-enabled Web Applications. In *Proceedings of the 2005 SIGMOD Conference, 2005*, pages 851–856. ACM, June 2005.
4. P. Brusilovsky. Methods and Techniques of Adaptive Hypermedia. *User Model. User-Adapt. Interact.*, 6(2-3):87–129, 1996.
5. F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Transactions on Database Systems*, 24(3):405–451, 1999.
6. S. Ceri, F. Daniel, and M. Matera. Extending WebML for Modeling Multi-Channel Context-Aware Web Applications. In *Proceedings of WISE'03 Workshops, Rome, Italy, December 12 -13, 2003*, pages 225–233. IEEE Press, 2003.
7. S. Ceri, P. Dolog, M. Matera, and W. Nejdl. Model-Driven Design of Web Applications with Client-Side Adaptation. In *Proceedings of ICWE 2004*, LNCS 3140, pp. 201-214, Springer Verlag.
8. S. Ceri, F. Daniel, M. Matera, and F. Facca. Model-driven Development of Context-Aware Web Applications. *ACM Transactions on Internet Technology (TOIT)*, Volume 7, Number 1, February 2007.
9. S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kauffmann, 2002.
10. S. Ceri, P. Fraternali, A. Bongio, S. Butti, R. Acerbis, M. Tagliasacchi, G. Toffetti, C. Conserva, R. Elli, F. Ciapessoni, and C. Greppi. Architectural Issues and Solutions in the Development of Data-Intensive Web Applications. In *Proceedings of CIDR 2003, January 2003, Asilomar, CA, USA*, 2003.
11. S. Ceri, P. Fraternali, and M. Matera. Conceptual Modeling of Data-Intensive Web Applications. *IEEE Internet Computing*, 6th(4):20–30, July-August 2002.
12. Chaeron Corporation. Chaeron GPS (Global Positioning System) Library. `http://www.chaeron.com/gps.html`, 2005.
13. F. Daniel, M. Matera, G. Pozzi. Combining Conceptual Modeling and Active Rules for the Design of Adaptive Web Applications. *Proceedings of the First International*

Workshop on Adaptation and Evolution in Web Systems Engineering - AEWSE'06, 2006.
14. R. De Virgilio, R. Torlone. A general methodology for context-aware data access. In *Proceedings of MobiDE'05*, pages 9–15.
15. M. Davis. Struts, an open-source MVC Implementation, February 2001. `http://www-106.ibm.com/developerworks/library/j-struts/?n-j-2151`.
16. P. De Bra, A. Aerts, B. Berden, B. de Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash. AHA! The Adaptive Hypermedia Architecture. In *Proceedings of HYPERTEXT'03*, pages 81–84, 2003.
17. A. K. Dey and G. D. Abowd. Towards a better Understanding of Context and Context-Awareness. In *Proceedings of CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness, The Hague, The Netherlands*, 2000.
18. Z. Fiala, M. Hinz, G.-J. Houben, and F. Frasincar. Design and Implementation of Component-Based Adaptive Web Presentations. In *Proceedings of SAC'04*, pages 1698–1704, 2004.
19. R. E. Filman, T. Elrad, S. Clarke, and M. Aksit. *Aspect-Oriented Software Development*. Addison-Wesley, 2004.
20. F. Frasincar, P. Barna, G.-J. Houben, and Z. Fiala. Adaptation and Reuse in Designing Web Information Systems. In *Proceedings of the International Conference on Information Technology - ITCC'04*, Track on Modern Web and Grid Systems, Volume 1, pages 387–391, IEEE Computer Society, 2004.
21. P. Fraternali. Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. *ACM Computing Surveys*, 31(3):227–263, September 1999.
22. F. Garzotto, P. Paolini, and D. Schwabe. HDM A Model-based Approach to Hypertext Application Design. *ACM Transactions on Information Systems*, 11(1):1–26, 1993.
23. M. Grossniklaus and M. C. Norrie. Information Concepts for Content Management. In *Proceedings of WISE'02 Workshops*, pages 150–159, 2002.
24. K., Henricksen and J. Indulska. Modelling and Using Imperfect Context Information. In *PerCom Workshops 2004*. 33–37.
25. K. Henricksen, J. Indulska and A. Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. In *Pervasive 2002*. 167–180.
26. Jin, Y., Decker, S., Wiederhold, G.: OntoWebber: Model-Driven Ontology-Based Web Site Management. In: The 1st International Semantic Web Working Symposium (SWWS'01), Stanford University, Stanford, CA, July 29-Aug 1, Springer Verlag (2001).
27. Laszlo Systems Inc. Openlaszlo - An XML Framework for Rich Internet Applications. Laszlo Systems Technology White Paper, July 2005.
28. H. Lei, D. M. Sow, J. S. Davis II, G. Banavar, M. Ebling, M. The Design and Applications of a Context Service. *Mobile Computing and Communications Review 6,* 4, 45–55, 2002.
29. N. Koch, A. Kraus, and R. Hennicker. The Authoring Process of the UML-based Web Engineering Approach. In D. Schwabe, editor, *First International Workshop on Web-oriented Software Technology (IWWOST01)*, 2001.
30. S. Long, R. Kooper, G. D. Abowd, and C. G. Atkeson. Rapid Prototyping of Mobile Context-Aware Applications: The CyberGuide Case Study. In *MOBICOM*, pages 97–107, 1996.
31. Macromedia Inc. Developing Rich Internet Applications with Macromedia MX 2004. Macromedia White Paper, August 2003.
32. Place Lab. A privacy-observant location system. `http://www.placelab.org`, 2006.

33. D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of CHI'99*, pages 434–441, 1999.
34. Schewe, K.D., Thalheim, B.: Reasoning about web information systems using story algebras. In: Advances in Databases and Information Systems (ADBIS 2004), volume 3255 of Lecture Notes in Computer Science, Springer Verlag (2004) 54–66.
35. A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven and W. V. de Velde. Advanced Interaction in Context. In *HUC*. 89–101, 1999.
36. D. Schwabe, G. Rossi, and S. D. J. Barbosa. Systematic Hypermedia Application Design with OOHDM. In *Proceedings of HYPERTEXT'96*, pages 116–128, New York, NY, USA, 1996. ACM Press.
37. M. Theodorakis, A. Analyti, P. Constantopoulos and N. Spyratos. Context in Information Bases. In *Proceedings of CoopIS'98*, pages 260–270, 1998.
38. R.Torlone et al. Front-end Methods and Tools for the Development of Adaptive Applications. In: Barbara Pernici (ed), Mobile Information Systems, Springer Verlag (2006), 209–246.
39. R. Vdovjak, F. Frasincar, G.-J. Houben, and P. Barna. Engineering Semantic Web Information Systems in HERA. *Journal of Web Engineering*, 2(1-2):3–26, 2003.
40. R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.
41. WebModels s.r.l. Webratio Site Development Studio. http://www.webratio.com, 2005.