

Chapter III

Context–Aware Applications for the Web: A Model–Driven Development Approach

Florian Daniel
University of Trento, Italy

ABSTRACT

Adaptivity (the runtime adaptation to user profile data) and context-awareness (the runtime adaptation to generic context data) have been gaining momentum in the field of Web engineering over the last years, especially in response to the ever growing demand for highly personalized services and applications coming from end users. Developing context-aware and adaptive Web applications requires addressing a few design concerns that are proper of such kind of applications and independent of the chosen modeling paradigm or programming language. In this chapter we characterize the design of context-aware Web applications, the authors describe a conceptual, model-driven development approach, and they show how the peculiarities of context-awareness require augmenting the expressive power of conceptual models in order to be able to express adaptive application behaviors.

INTRODUCTION

The evolution of the Information Technology in the last years has seen the World Wide Web transforming from a read-only hypertext media into a full-fledged, multi-channel and multi-service application delivery platform. Current advances in communication and network technologies are changing the way people interact with Web ap-

plications. 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. More and more users themselves ask for services and applications highly tailored to their individual requirements and, especially due to the increasing affordability of new and powerful mobile communication devices,

they also begin to appreciate the availability of ubiquitous access. In order to cope with the growing demand for novel, user-centric application features, such as adaptivity and context-awareness, appropriate development methods for Web applications are required.

Adaptivity is increasingly gaining momentum in the context of modern software systems. Runtime adaptivity provides highly flexible and responsive means for the customization of contents and services with respect to the user's identity. Varying device characteristics in mobile and multi-channel computing environments can be adequately taken into account and leveraged by means of adaptive software designs, whose development is facilitated by the availability of standardized communication protocols (e.g. HTTP) and markup languages (e.g. HTML or WML), supported by most of today's mobile devices. Multi-channel deployment does no longer require completely different, parallel design approaches and rather represents a presentation issue on top of unified engineering solutions.

But adaptivity may also enable an application to take into account a wider range of properties describing the interaction between the user and the application, thus paving the way for context-awareness. *Context-awareness* (Dey & Abowd, 2000; Schilit & Theimer, 1994) is often seen as recently emerged research field in information technology and in particular in the domain of the Web. From the perspective of application front-end development it can however be interpreted as natural evolution of personalization and adaptivity, addressing not only the user's identity and preferences, but also his/her usage environment. Personalization has already demonstrated its benefits for both users and content providers and has been commonly recognized as fundamental factor for augmenting the efficacy of the overall communication of contents. Context-awareness goes one step further in the same direction, aiming at enhancing the application's usefulness

and efficacy by combining personalization and adaptivity based on an application-specific set of properties (the context) that may affect the execution of the application.

In this chapter, we focus on the development of context-aware applications for the Web and, in particular, we describe a *model-driven* development method that allows developers to approach the problem at a level of abstraction that enables him/her to focus on the real design challenges of such class of applications, leaving low-level implementation concerns to supporting CASE (Computer-Aided Software Engineering) tools. Considering that software systems are continuously getting more complex and difficult to maintain – partly due to the previously described requirements –, we believe that efficient abstraction mechanisms and design processes, such as those provided by visual, model-driven design methods, are becoming crucial. The focus on essential design issues and the ease of reuse in model-driven design methods may significantly accelerate the overall design process. As we will show in this chapter, starting from application models, code generation techniques may then provide for the automatic generation of application code or templates, thus assuring the fast production of consistent and high quality implementations.

MOTIVATING EXAMPLES

Active application features, such as context-aware or adaptive behaviors, may augment the effectiveness of interactions and the efficiency of resource consumption in all those situations where services and contents offered by an application strongly depend on environmental situations, users' abilities or disabilities, or the state or health of a software system. For example, typical applications demanding for active features and adaptivity are:

- *Adaptive personalization.* User profile attributes for personalization purposes may present different levels of variability in time. Profile properties may be static in nature (e.g. the name of a user), slowly changing (e.g. profile data derived from a user's browsing behavior) or even fast changing (e.g. the pulse frequency of a patient). Adaptive personalization mechanisms that take into account such profile peculiarities could allow systems to go beyond the common and static tailoring of services and contents.
- *Interaction-enabling functionalities.* Context could as well consider handicaps or physical disabilities of users, such as vision problems, blindness or paralysis, to adapt the application accordingly and to provide alternative and better suited interaction mechanisms and modalities. Adaptivity could thus provide functionalities enabling handicapped users to properly interact with applications, thus fostering the accessibility of applications.
- *Effective content delivery.* In general, whatever context data may be leveraged to provide appropriate contents and program features at the right time, priority, and emphasis. For example, specifically targeted special offers could be advertised and directed more effectively, presentation properties could be adapted to varying luminosity conditions for better readability, etc. Adaptive or context-aware extensions could thus enhance the overall effectiveness of applications by adapting individual application elements to varying users or usages of the application.
- *Delivery of context as content.* Applications may depend intrinsically and in a structural manner from context data. Location-aware applications, such as city map services or navigation systems, treat position data as core contents of the application and adapt to them, supported by proper localization mechanisms. To such kind of applications, the use of context data represents a functional requirement, rather than an optional feature.
- *Exception handling.* Critical events during the execution of a software system may raise exceptions and require prompt reactions being performed. Process-based or workflow-driven applications, for example, represent a typical class of applications that constantly have to cope with exceptional situations in order to guarantee the consistent termination of a running process. Here, adaptive or context-aware mechanisms could be leveraged to capture respective application events and to enact the pieces of application logic that are necessary to handle the exceptional situation.
- *Production and control systems.* Critical production or control systems may require, for example, highly specific sensing and alerting mechanisms to prevent production losses or product quality degradations. Context-awareness could facilitate the timeliness of reactions and the efficient handling of dangerous situations, but also proactive maintenance approaches, such as those adopted in a steadily growing number of hardware/software systems, may be achieved.
- *Self-healing software systems.* Autonomic or self-healing software systems elevate the idea of proactive maintenance from hardware to software systems and aim at the creation of computing systems that are able to configure, tune, and even repair themselves. Proactive and adaptive capabilities in this context are an essential feature.

REFERENCE SCENARIO

To exemplify the concepts introduced in this chapter and to better convey the underlying ideas, step by step we will show how we developed one

of our demonstration prototypes, the PoliTour application. The application runs on a PDA with wireless Internet access and enables visitors to Politecnico di Milano, Italy, to obtain location-aware campus details (i.e. information about roads and buildings) while walking through the campus. If a user is about to leave the WiFi-covered area of the campus, an alert message is shown.

CONTEXT-AWARENESS AND WEB APPLICATIONS

Due to a lack of appropriate technologies and concepts, for a long time context-awareness has not been considered suited to the domain of the Web. Web technologies (both hardware and software) are however continuously evolving and the attitude toward reactive and context-aware behaviors in Web applications is changing. As a matter of fact, support for a multitude of non-functional requirements, whose inadequate coverage prevented the adoption of Web technologies for the implementation of reactive applications, has now been developed. Just to mention a few:

- The *reliability* of data communications has been considerably enhanced along both the software and the hardware dimension. The introduction of reliable messaging techniques (e.g. digital certificates or the WS-Reliability specification) provides for trustworthy communications on top of standard Web protocols, such as HTTP or SOAP. The success of fiber optics – as an example of hardware evolution – has allowed the Ethernet protocol (typically used in the Web) even to enter industrial production environments, where the high electromagnetic interferences that exist in the presence of high-voltage machineries practically prohibited the use of conventional, unreliable network technologies.

- The *pervasiveness* and *availability* of Web applications is continuously growing due to the introduction of novel networking technologies, such as ADSL (Asynchronous Digital Subscriber Line) or fiber optics for home and office users and WiFi and 3rd generation mobile telephony technologies (e.g. UMTS, GPRS, EDGE) for mobile users.
- Web applications have proved a high *scalability* (it suffices to think about certain portal applications that serve millions of users every day), facilitated *maintainability* and high *cost efficiency*.

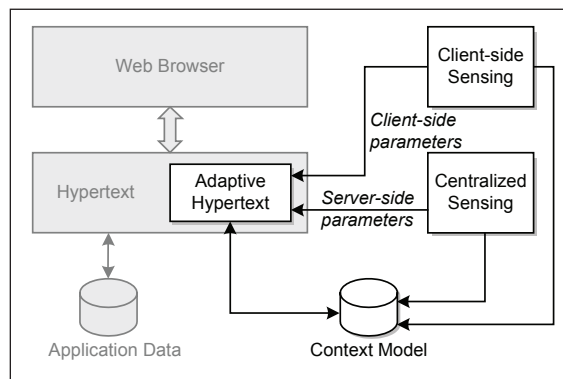
Provided that technological advances enable and facilitate the development of adaptive Web applications, it is important to recognize that context-awareness, rather than being a mere technological concern, represents a true *design* issue. In the following, we will thus focus on the typical design concerns in the development of context-aware Web applications.

Enabling Context-Awareness in the Web

Developing context-aware applications for the Web demands some characteristic architectural components, in order to support adaptations to context. Figure 1 proposes a possible functional architecture that extends the traditional architecture of Web applications with components aimed at supporting the acquisition, storage, and use of context data.

The typical context-aware application's data source includes both the application data (i.e. the business objects that characterize the application domain and the user) and the context model, which offers at any moment an up-to-date representation of the context state. The context model captures all the context-characterizing properties and enables the system to adapt to

Figure 1. Context data in context-aware Web applications. Gray shaded boxes correspond to conventional, non-adaptive parts, white boxes correspond to extensions required to support context-awareness.



changes thereof, assuming that such changes may demand for proper reactions by the application. An application typically consists of adaptive (i.e. context-aware) and non-adaptive parts; we call the former adaptive hypertext. The pages of the adaptive hypertext present some form of adaptive behavior, i.e. they are able to react to changes in the context, while pages of the non-adaptive hypertext do not present any adaptive behavior. To decide which adaptation is required – if any –, the adaptive hypertext makes use of context data during the rendering of hypertext pages. Context data needs to be sensed (e.g. by means of suitable instruments, such as GPS positioning systems, thermometers, or similar) and communicated to the Web server that hosts the application, in order to be processed.

The above architecture allows for three main communication mechanisms to pass context data from the sensing devices to the application: (i) as parameters sensed at the *client side* and sent to the application (e.g. GPS position data); (ii) as *server-side* parameters (i.e. HTTP session variables)

provided by a centralized sensing infrastructure (e.g. system usage data); and (iii) by means of direct updates of the *context model*. Typically, client-side parameters are generated by client-side sensing solutions, server-side parameters are filled by centralized sensing solutions, and database updates may be performed by both.

Context-awareness in Web applications therefore requires addressing the following issues:

- *Context data modeling.* Context properties that are relevant for the provisioning of the context-aware behaviors of the application must be identified and represented in an application-accessible format. The result of this task is the context model that can be queried for adaptation purposes.
- *Modeling of adaptive application behaviors.* Starting from the context model, adaptation operations need to be defined in order to react to situations demanding for adaptation. That is, detected changes to the context data are translated into visible effects or operations that aim at augmenting the effectiveness and usability of the application.
- *Context model management.* The context model only captures the static aspect of context data, i.e. their structure; in order to also capture the dynamics of context data, and hence to be able to trigger adaptive behaviors, we also need to:
 - *Acquire context data* by means of measures of real-world, physical properties, corresponding to the properties of the context model. The so acquired data are then fed into the context model, so as to keep the context model up to date.
 - *Monitor context data* to detect those variations in context data that trigger adaptivity. Relevant variations are used to enact the adaptation operations in the adaptive hypertext, thus causing an automatic, adaptive behavior of the Web application.

While the definition of the context model and the monitoring of context data can easily be assisted by proper context modeling methods and a proper runtime framework providing basic monitoring functions, it is not as easy to assist designers in the development of suitable context acquisition (i.e. sensing) infrastructures. In fact, the former two activities can be generalized beyond the needs of individual applications, while the design of sensing infrastructures remains tightly coupled with individual application requirements and technological choices. The exact development of sensing infrastructures is thus out of the scope of this chapter.

Context-Aware Behaviors in Web Applications

But what exactly does it mean to adapt a Web application or to react to context? Starting from the work by Brusilovsky (1996) on adaptive hypermedia systems, in context-aware Web applications, adaptive behaviors may affect:

- *Contents and services* delivered by the accessed pages: the application may autonomously chose contents or services based on changing context data.
- *The navigation*: the application may perform automatic navigation actions on behalf of the user toward pages that better suit the current context conditions.
- *The whole hypertext structure*: the application may choose to apply coarse-grained adaptations (e.g. to the layout of the application), for example to react to changes of the user's device, role, or activity within a multi-channel, mobile environment.
- *Presentation properties*: the application may apply more fine-grained adjustments to the application's appearance (e.g. to style properties or fonts in use).
- *Generic operations*: the application may decide to enact generic operations in the

background, e.g. to log specific application events or to interact with external applications.

In this chapter, we will describe how these behaviors have been realized in the model-driven design method WebML and how the resulting extended version of the method can be leveraged for the development of context-aware applications. Before proceeding with the discussion, it is thus appropriate to shortly introduce the WebML development method, which will serve as reference throughout this chapter.

The Web Modeling Language (WebML)

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 (Ceri et al., 2002).

WebML application design starts with the specification of a *data schema*, expressing the organization of the application contents by means of well established data models, such as the Entity-Relationship model or the UML class diagram. On top of such data schema, WebML design then proceeds with the specification of a so-called *hypertext model*, which describes 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* to constitute site views. Besides representing user navigation, links between units also specify the transportation of parameters that can be used by the destination unit in its selector condition. Some WebML units also support the specification of content management operations. Standard operations are creating, deleting or modifying an instance of an entity or adding or dropping a relationship between two instances; custom units may be defined. Finally, WebML also allows the management of *session parameters*; parameters can be set and consumed through proper units.

In addition to the visual representation, WebML also comes with an XML-based, textual representation, which allows one to specify additional detailed properties, not conveniently expressible in the graphic notation. The availability of the XML specification enables the automatic generation of the application code (Web Models, 2008), comprising rendering formats like HTML (which is the standard choice for deployment) or WML. For a detailed description of WebML, the interested reader is referred to (Ceri et al., 2002).

MODELING DATA FOR CONTEXT-AWARE WEB APPLICATIONS

Context data can derive from several sources integrating sensed, user-supplied, and derived information (Henricksen, 2004; Henricksen,

2002). While user-supplied data are generally reliable and tend to be static, sensed data are typically highly dynamic and can be unreliable due to noise or sensor errors. The problem of unreliability has been addressed in literature for example by associating context information with quality data (Lei, 2002). Although we recognize the importance of reliable context data, in this work we rather concentrate on the exploitation of context in the design of Web applications. For simplicity, throughout this chapter we thus consider sensed data as trustworthy.

Characterizing Context Data

The main goal of context modeling is the formalization and abstraction of the context properties that affect the application. In this regard, a first characteristic distinguishing context properties is the distinction between physical and logical context. We call *physical* context those properties that are immediate representations of sensed, physical quantities (e.g. the values of an analog/digital converter), and *logical* context those properties that enrich physical context with semantics and additional abstractions of the raw sensed data (e.g. the city corresponding to physical longitude and latitude values).

A second characteristic affecting the structure of the context model is the *persistence* of context properties in the system, i.e. the property that expresses whether individual context properties represent persistent data or volatile data. *Persistent* data need to be stored in the application's data source and therefore require proper data entities being modeled as part of the context model, while *volatile* data do not need any storage and can thus be omitted from the context model. The context model therefore only captures persistent context data (indeed, in WebML the context-model is part of the database underlying the application).

Starting from these two characteristics and from the reference architecture introduced in Figure 1, Figure 2 summarizes the resulting

Figure 2. Persistence of physical and logical context data.

Context Abstraction	Logical Context	/	Logical context stored as data in the context model
	Physical Context	Physical context for one-time consumption only	Physical context stored for context sharing or tracking purposes
		volatile	persistent
		Persistency	

characterization of context data:

- *Volatile physical context.* Context data communicated via client-side parameters or via server-side session parameters represent volatile data. They are immediately available during the execution of the application, independently of the underlying context model. Volatile context data are not enclosed in the context model; they might however be used during page computation to adapt the application.
- *Persistent physical context.* Context data sharing (e.g. between members of a same group) or tracking (e.g. to derive differential context properties or to keep a context history) typically require the persistent storage of data. Persistent physical context data are thus included in the context model and updated according to their dynamics.
- *Persistent logical context.* Logical context data is stored as data in the context model, so as to enable the data-driven transformation of physical context into logical context. Logical context data are typically static, as they provide abstractions of physical context; dynamic updates and/or extensions can, however, be supported as well.

Physical and logical context data therefore coexist in the application's data source. This coexistence typically requires a transformation or mapping between raw data and information that can directly be used when specifying hypertext schemas. Consistently with the data-driven approach that characterizes WebML, we propose a formalization of such transformation at the data level by means of suitable associations between data entities representing physical and logical context data, respectively. Although technically legal, we do not expect the use of *volatile logical context*, as volatile context data typically represents sensed raw context data.

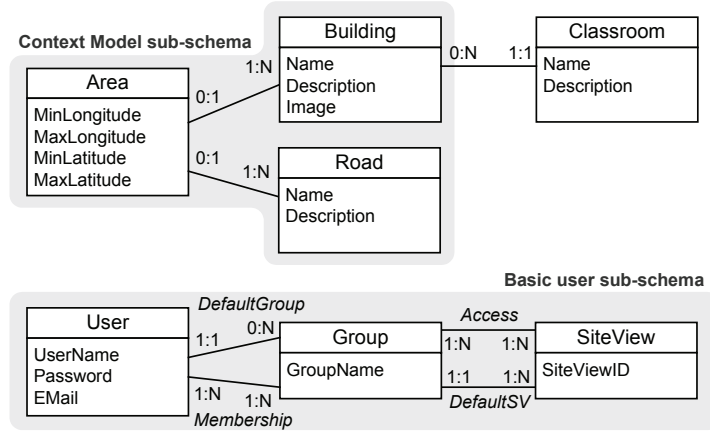
It is worth noting that even though there are several properties commonly regarded as *context attributes* (e.g. position, time, or device characteristics), there exists no universal context model that applies to all kinds of applications. For this reason, also in this chapter we do not prescribe any precise, rigid context model for WebML applications; we rather introduce some WebML-specific modeling guidelines that enable the designer to provide context-aware applications with suitable context meta-data.

Example Data Schema for Adaptation in WebML

Let's consider the PoliTour application shortly discussed in the introduction. Figure 3 illustrates a possible Entity-Relationship diagram with basic user profile data and context data, grouped in the figure into so-called sub-schemas:

- *User profile sub-schema.* Users, groups, and site views are represented as “first-class citizens” in the application data source, as required by the WebML design process. The entity User provides a basic profile of the application's users, the entity Group associates access rights to users (i.e. a role), and the

Figure 3. Adaptation-triggering data in WebML applications, partitioned into basic user sub-schema, personalization sub-schema and context sub-schema.



entity Site View contains the site views that may be accessed by the members of a group. The relationship Membership expresses that users may belong to multiple groups, which in turn cluster multiple users. The relationship DefaultGroup connects a user to his/her default role and, when logging into the application, the relationship DefaultSV allows the application to forward the user to his/her default group's default site view. The relationship Access expresses which site views a specific group is allowed to access; this relationship is required as varying context conditions may require different interaction and navigation structures for a same group. In this way, depending on the context state, the application is able to determine the most appropriate site view and to forward the user accordingly.

- *Context model sub-schema.* The context model of the application is represented by the entities Area, Building, and Road, which all provide logical context data. The actual

GPS position data used for delivering the location-aware guide through the Politecnico campus (i.e. longitude and latitude) and the signal strength of the WiFi connection are not part of the context model in the application's data source; in developing the PoliTour application, we will handle such as volatile context data. Starting from the physically sensed data, the entity Area allows the application to identify a geographical area inside the campus; an area is then associated either with a Building or a Road, meaning that starting from the user's position we can identify whether he/she is located close to a building or rather walking through one of the roads in the campus.

- *Application data.* The remaining entity Classroom represents application data that are not part of the context model. This means that from a building it is possible to access the list of classrooms of the building, but there are no adaptive behaviors associated with the entity Classroom.

MODELING CONTEXT-AWARE HYPERTEXTS

While the first step of the WebML design method, i.e. data modeling, does not require any extension of the modeling primitives for capturing context data (the standard Entity-Relationship primitives suffice), WebML hypertext modeling does require a few model extensions to express adaptivity concerns. Next we therefore introduce the new concepts and primitives that have been developed to express adaptive behaviors, and we clarify how different adaptivity policies can be used to enact adaptations.

Context-Aware Pages and Containers

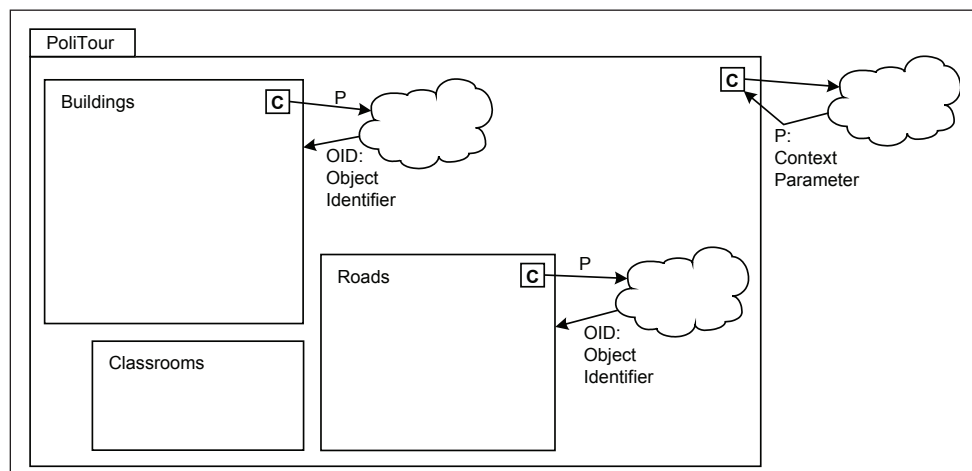
Our basic assumption in the modeling of context-aware hypertexts is that context-awareness or adaptivity is a property to be associated only to some *pages* of an application (the *adaptive hypertext*), not necessarily to the application as a

whole. Location-aware applications, for example, adapt core contents to the position of a user, and so-called “access pages” (e.g. containing categories or lists) typically are not affected by the context of use.

As can be seen in Figure 4, we tag context-aware pages with a C-label (standing for *context-aware*) to distinguish them from conventional pages. The label indicates that an adaptivity rule (stylized as a cloud) is associated with the page and that during the execution of the application this logic must be taken into account when computing the page. Specifically, Figure 4 states that pages Buildings and Roads are context-aware, while the page Classrooms does not present any adaptive behavior.

There might also be the need for adaptivity rules with effects that spread over multiple pages. For this purpose, we exploit the hierarchical structure of hypertexts; that is, we allow the definition of context-aware *containers* (i.e. *site views* and *areas*, in terms of WebML). This allows the designer to insulate and to specify only

Figure 4. WebML hypertext schema with one context-aware site view and two context-aware pages. The parameter *P* exemplifies the propagation of reusable context data by hierarchically passing context parameters from an outer area to an inner page.



once adaptivity rules that are common to multiple C-pages inside a container and thus to reduce the redundancy of the schema. Adaptivity rules associated to containers and pages are evaluated recursively, starting from the outermost container and ending with the actual pages. The site view PoliTour in Figure 4 is context-aware; we will see later on why.

Localized and Sparse Adaptivity Rules

The adaptivity rules attached to the context-aware pages and containers in Figure 4 represent the actual adaptivity logic (i.e. the set of adaptivity actions to be performed). The adaptivity logic is external to the page or container, and the chain of adaptivity actions it clusters is kept separate from the page or container specification. The aim is to highlight the two different logics deriving from the role played by pages/containers and adaptivity operations: while the former act as *providers* of contents and services, the latter act as *modifiers* of such contents and services.

Adaptivity actions attached to a C-page typically present effects that are visible in the page they are attached to. The notion of context-aware page and adaptation logic therefore defines what we call a *localized adaptivity rule*: the scope of a localized adaptivity rule is strictly coupled with a fixed set of hypertext pages, where “scope” refers to those (adaptive) pages to which the page’s adaptivity actions are associated.

The notion of context-aware container allows us to define *sparse adaptivity rules*: we talk about *sparse* adaptivity rules in those cases, where adaptivity actions are associated to containers that contain multiple pages; the scope of such actions spans a set of pages, more precisely, all context-aware pages in the container. Coming back to the PoliTour application sketched in Figure 4, we can thus associate the logic to interpret the signal strength of the WiFi connection to the

pages Buildings and Roads by applying the logic to the site view as a whole.

Parameter Passing

Adaptivity logic is associated to a page by means of a directed arrow, i.e. a link exiting the C-label. This link ensures the communication between the page logic and the adaptivity logic: it may transport parameters deriving from page contents, which may be used to compute the specified actions; in turn, a link from the adaptivity logic to the page may transport context parameters or generic values that might be required to perform the final adaptation during page computation.

But Figure 4 also illustrates the possibility of *hierarchically* passing parameters from an outer container to an inner one. More precisely, if the evaluation of outer adaptivity logic produces results to be reused at an inner level, as it might happen in the case of context parameters, it passes such values back to the C-label that activated the computation of the logic. Subsequently, such parameters can then be “consumed” by adaptivity logics of the inner levels. As for context-aware pages, parameter passing from a container to its adaptivity logic occurs through the logic-activating link. Links exiting the last evaluated logic, i.e. at the end of the last adaptivity action, might carry parameter values for the computation of units inside a page.

Typical actions to be specified at the container level are the acquisition of fresh context data and the updating of the context model, e.g. if the data are to be shared among multiple users or if a history of context data is to be tracked. Hence, especially if persistent context data are adopted, we propose two levels for adaptivity actions:

- *Actions for context model management*, addressing operations for context data acquisition and context model updating, should be associated with outer containers

(site views or areas) and are inherited by inner containers (areas or pages). These adaptivity actions need to be executed prior to the execution of any other action possibly specified in an inner context cloud, as such “internal” actions could depend on context data acquired and stored in the context model through “external” actions.

- *Actions for hypertext adaptivity*, defining the rules for page and navigation adaptation (and possibly depending on persistent context data), should be associated with C-pages.

Specifying Adaptivity Logics

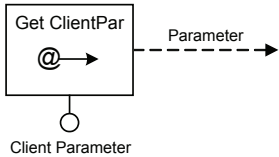
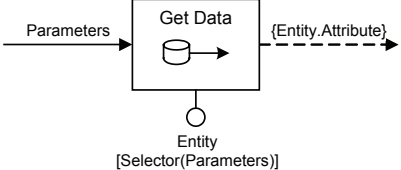
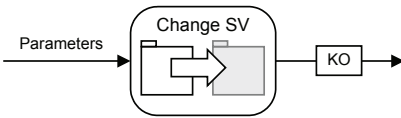

The main novelties for modeling context-aware pages reside in the specification of adaptivity rules by means of WebML constructs. In the

following, we introduce the new WebML modeling concepts that ensure full coverage for the specification of context model management and hypertext adaptation logics. The new primitives allow designers to visually specify actions for acquiring and updating context data and to define adaptivity actions.

Managing Context Data

In order to support adaptivity with respect context, the application must be able to acquire and manage context data according to the mechanisms illustrated in Figure 1. For this purpose, some new WebML operations have been defined, which, together with the already available operations, provide the necessary primitives for:

Figure 5. WebML units that have been defined for the specification of adaptivity actions.

Visual Notation	Description
	<p>Input: no input</p> <p>Source Parameter: parameters generated at the client side</p> <p>Output: parameter value</p>
	<p>Input: parameters for selector condition evaluation</p> <p>Source Entity: database entity from which to extract the data rows to be filtered by the selector condition</p> <p>Output: (set of) parameters or attributes retrieved</p>
	<p>Input: identifiers of target site view and target page, last user selections, global parameters, context parameters</p> <p>Output (KO-link): no output</p>
	<p>Input: filename of CSS file to be associated to current site view</p> <p>Output: no output</p>

- *Specifying the acquisition of fresh context data through client-side parameters.* A new Get ClientParameter unit (see Figure 5) has been defined to support the retrieval of parameters generated at the client side and communicated back to the application via client-side parameters (e.g. parameter-value pairs attached to the page request query string).
- *Specifying the acquisition of fresh context data through server-side parameters.* Context data directly made available as HTTP session parameters can be accessed by means of conventional WebML Get units (Ceri et al., 2002).
- *Specifying the acquisition of context data from the context model.* The execution of adaptivity actions may require the retrieval and evaluation of context meta-data, for example, in situations where certain data are just needed to evaluate condition expressions. For this purpose, a so-called Get Data unit (see Figure 5) has been introduced, enabling the retrieval of values (both scalars and sets) from the data source according to a selector condition. The semantics of the Get Data unit is similar to the one of content publishing units (Ceri et al., 2002), with the only difference that data retrieved from the data source are not published in hypertexts, but just used as input for units or operations.
- *Updating the context model.* Once fresh context parameters have been retrieved, they can be used to update the context model at data level. This action consists in modifying values previously stored in the data source. In WebML, this is already facilitated by operation units (Ceri et al., 2002) providing support for the most common database management operations (e.g., modify, insert, delete).

Evaluating Conditions

The execution of adaptivity actions may be subject to the evaluation of some *conditions*, refining the triggering logic for context clouds. The most recurrent pattern consists in evaluating whether context changes demand for adaptation. The evaluation of conditions is specified by means of two control structures, represented by the If and Switch operation units, which have been introduced for workflow modeling in WebML (Brambilla et al., 2003).

Executing Adaptivity Actions

Once the current context state has been determined, and possible conditions have been evaluated, adaptivity actions can be performed to adapt the page contents, the navigation, the current site view structure, and/or presentation style properties. These actions are specified as follows:

- *Adapting Page Contents.* Page contents are adapted by means of proper data selectors, whose definition is based on context parameters retrieved from the context model or newly computed within the page's context logic. The use of parameterized selectors allows for both *filtering* data items with respect to the current context state and conditionally *including/excluding* (i.e. showing/hiding) individual content units.
- *Adapting Navigation.* In some cases, the effect of condition evaluation within the context cloud can be an automatic, i.e. context-triggered, navigation action, causing the redirection of the user to a different page. The specification of context-triggered navigations just requires connecting one of the links exiting the adaptivity logic of the page to an arbitrary destination page of the hypertext. Therefore, links exiting the con-

text cloud and directed to other pages than the adaptivity logic's source page represent automatic navigation actions.

- *Adapting the Site View.* In some cases, a context-triggered switch toward a different site view may be required. Changes in the interaction context may in fact ask for a coarse-grained restructuring of the whole hypertext, for example because the user device has changed, or because the user shifted to a different activity. To switch between different site views, we have introduced a Change Site View unit (see Figure 5), which takes in input the identifiers of the target site view and the target page, to be visualized in case a switch toward the specified site view is required. In order to support “contextual” switching, the input link also transports parameters characterizing the current state of interaction, i.e.:
 1. The input parameters of the source page, which represent the last selections operated by the user;
 2. Global parameters, representing session data (e.g. user identifier and group identifier), as well as past user selections that have been used for the computation of the current page;
 3. Client-side and server-side context parameters retrieved during the latest performed data acquisition cycle and characterizing the current context state.
- *Adapting Presentation Style.* Sometimes context changes may require only fine-grained adaptations of presentation properties (e.g. due to varying luminosity conditions), not a complete restructuring of the overall hypertext. We have defined a Change Style unit for dynamically assigning presentation style properties (see Figure 5). Style properties are collected in proper .css (Cascaded Style Sheet) files, and the

unit enables the application to change its associated style sheet at runtime.

- *Enacting generic operations.* The context-triggered invocation of generic operations or, for instance, external Web services can easily be specified by placing the respective WebML operation unit into the page's adaptivity logic and by providing the unit with the necessary input parameters.

Triggering Adaptivity Rules

But *when* do we enact an adaptivity rule? In this regard, it is possible to define two different *adaptivity policies* for context-aware pages, assigning different priorities to users and context:

- *Deferred Adaptivity:* the *user* is granted the highest priority. Therefore, after the user has entered the page and the page has been rendered according to the user's selections, the page's adaptivity logic is evaluated at periodic time intervals, enabling the application to possibly adapt the already rendered page. Periodically evaluating the adaptivity logic means periodically refreshing the page visualized in the browser.
- *Immediate Adaptivity:* *context* is granted the highest priority. The page's adaptivity logic is evaluated each time the page is accessed, being the access due to the user or to the periodic refresh of the page. This means that the page is subject to adaptation each time it is rendered, even at the first time the page is accessed by the user.

Consider for example our PoliTour guide that shows contents about the buildings and roads in the Politecnico campus. At a given point, the user might want to get information about a specific building located in a road that is not related to his/her current position; such a preference is typically expressed by selecting a link to that building from a list. With a deferred policy, the

requested page shows the building 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 subject to adaptivity and the contents are adapted to the user's location. With an immediate policy, context is granted higher priority with respect to the user and, thus, the user's request for the building would be overwritten by the context and the application would show the building or road associated to the user's current location, discarding the user's selection.

Note that in addition to these adaptivity policies, we recognize that there may be situations that demand for an explicit control of the adaptation dynamics through the user. Therefore, should for example a user temporarily not be interested in having the contents adapted to his/her location, he/she can simply disable/enable adaptivity at will. In WebML, the adaptivity policy for context-aware pages and containers is declared by means of the `Adaptivity_Policy` property of context-aware pages and containers.

Adaptivity policies can also be associated to context-aware containers. When a C-page is requested, also the possible context clouds of its containers are evaluated recursively (from the outermost one to the innermost one), according to the adaptivity policy associated to each container. In general, a container's adaptivity policy is independent of the policy of inner containers and pages (if not, this must be taken into account by designers when associating policies to containers and pages). Therefore, it may happen that the actions in a container's context cloud are evaluated immediately, even if the actions associated to inner containers or pages adopt a deferred evaluation, or vice-versa. If, for example, the adaptivity actions associated to the container serve for tracking a context history, they could require an immediate policy, while inner adaptivity actions keep their deferred policy for front-end adaptations. The hierarchical definition of context clouds may

therefore also be considered a facility to achieve different "layers" of adaptivity actions.

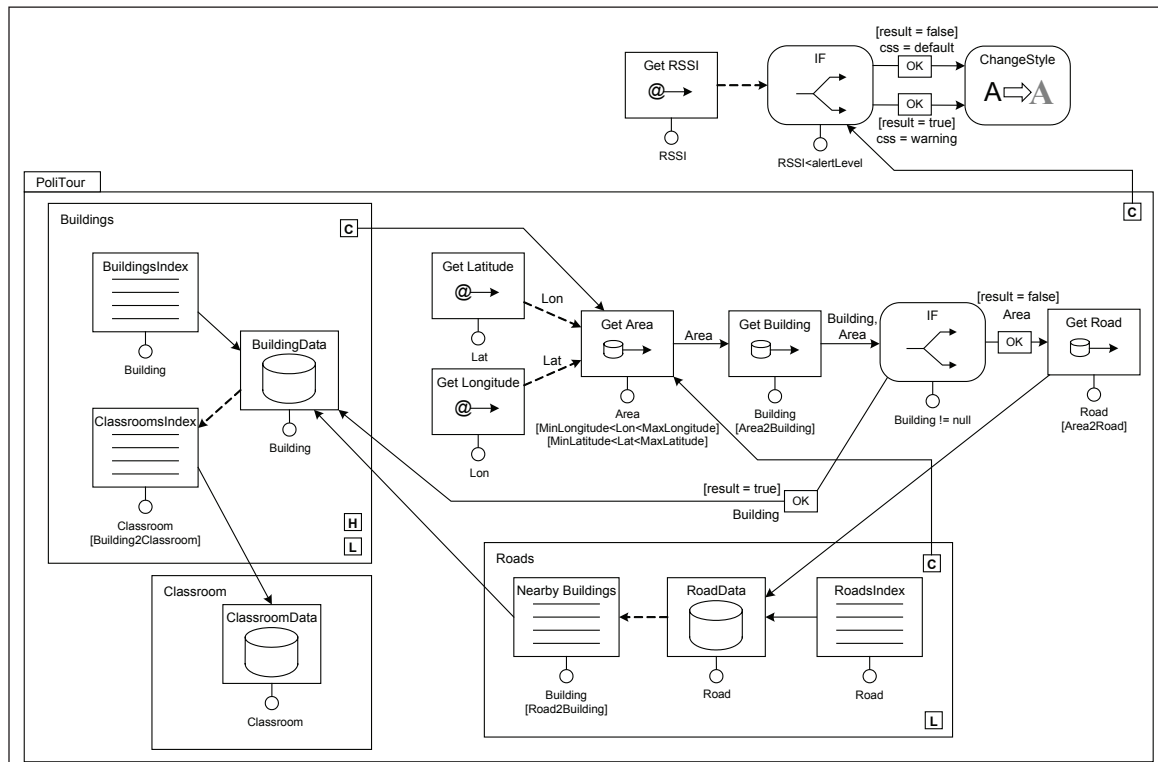
In our approach, we assume *deferred* adaptivity as default policy. This choice aims at minimizing application behaviors that might be perceived as invasive or annoying by users and has been experienced as the most natural for modeling adaptation. However, the *immediate* policy could be needed for handling exceptional situations, as in such cases the timely reaction to context changes could be more important than following the user's indications. We therefore, in general, recommend the selection of the adaptivity policy that is appropriate to the application requirements and that is able to minimize the application behaviors that could be perceived as invasive or annoying by the users. In order to choose the right adaptivity policy for an adaptive page, a developer therefore needs to predict what kind of adaptive behavior a user will expect when accessing that page.

Example Hypertext Model

Figure 6 shows the adaptive WebML hypertext model of the PoliTour application. The figure provides a refinement of the coarse hypertext model introduced in Figure 4 and details the internals of pages and adaptivity logics.

The pages Buildings and Roads share the same adaptivity logic providing location-awareness to the displayed contents. The logic starts with two `Get ClientParameter` units accessing the user's longitude and latitude, which are then used by the `Get Area` unit to associate a logical area to the user's position. A further `Get Data` unit (the `Get Building` unit) then tries to retrieve a building for the identified area. If a building could be retrieved, the `If` unit sends the user to the Buildings page, providing updated page parameters. If instead no building could be retrieved (e.g. because the user is located in the center of a road or not close enough to a building), the `If` unit forwards the Area identifier to the `Get Road` unit, which retrieves the road associated to the current position.

Figure 6. Hypertext model of the PoliTour application leveraging volatile context data.



Therefore, if the user views the page Buildings while walking around the campus, the application automatically updates the contents published each time a new building can be found. If only the road can be identified, the application performs an automatic navigation action toward the Roads page, where the described adaptive behavior starts again, possibly causing the adaptation of contents or automatic navigation actions. Only if the user navigates to page Classroom, no adaptations are performed, as this page is not tagged as context-aware.

The adaptivity actions associated to the surrounding site view specify how to alert users who are about to leave the WiFi-covered area. The Get RSSI unit accesses the volatile RSSI

parameter sensed at the client side, and the If unit compares the retrieved value with a predefined level (alertLevel), below of which the connectivity is considered low. In case of low connectivity, the style sheet warning is adopted; otherwise, the default style sheet is adopted. We therefore model the alert of low connectivity conditions by means of a Change Style unit: under low connectivity conditions the application is rendered with a red background, under normal conditions the application is rendered with a gray background.

We recall that actions associated to containers are evaluated before any action at the page level is started. Hence, in Figure 6 the actions associated to the site view are executed before the actions associated to the pages Buildings and Roads.

RUNTIME CONTEXT MODEL MANAGEMENT

In order to manifest context-aware behaviors, the application must be equipped with the capability to monitor the context state and to trigger adaptivity actions, if required. The standard HTTP protocol underlying most of today's Web applications implements a strict *pull* paradigm, in which computations can only occur in response to client-side generated page requests. Therefore, in the classical Web architecture, lacking proper push mechanisms, context monitoring can occur only when a page is computed, i.e. when a respective page request reaches the Web server. Three main solutions can be adopted to trigger the evaluation of adaptivity rules: (i) context evaluation on user-generated page requests, (ii) periodical, automatic refreshes of viewed pages to enable context evaluation, and (iii) active context evaluation to trigger adaptivity in real time. The first solution is not able to cope with the dynamic nature of context. The periodic refresh of context-aware pages provides a way to ensure the update of the page even in absence of explicit user actions enabling the re-computation of the page. In the following, we will show an active mechanism for triggering adaptivity, which operates independently of the user in the background and comes close to the real-time triggering solution.

In absence of dedicated server-side *push* mechanisms for delivering updated pages, the HTML http-equiv META-option or JavaScript, JavaApplets, and Flash scripts, provide valuable client-side mechanisms to approximate the required active behavior. The approximation is based on periodic HTTP requests toward the application server, which are operated in the background and may serve a twofold purpose:

- On the one hand, they provide the necessary polling mechanism to query the context model and trigger the adaptivity rule attached to the page.

- On the other hand, generating page requests allows the client to transmit client-side sensed data, thus enabling the communication of context data to the application server.

Context-aware pages are therefore also characterized by an individual *refresh interval*, which can be specified as property (Refresh_Interval) of the page in the XML representation of the WebML model. Differently from C-pages, a container does not require the specification of any polling interval, which is instead derived from the interval associated to the currently viewed C-page of the container.

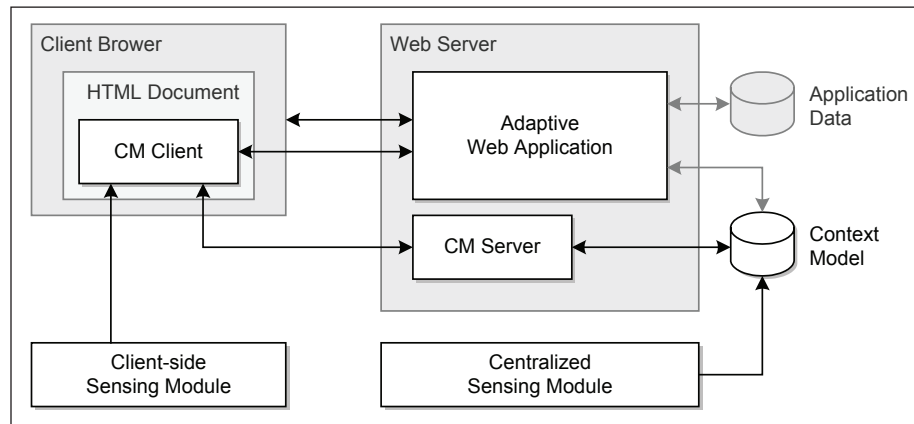
Context Monitoring

Context monitoring in the background (i.e. without the user observing any unwanted rendering activity) enables the application to limit the use of the refresh to those situations that really ask for adaptation and to perform context monitoring without any visual effect for users.

Figure 7 shows a functional architecture for adaptive Web applications that extends the described architecture of WebML applications (see Figure 1) with a new client-server module, called *Context Monitor* (CM), providing the necessary context monitoring logic. As further depicted by the figure, in case of client-side context sensing, the CM module also enables the communication of client-side sensed context parameters, which could be required at the server side to evaluate context changes and/or conditions over context parameters.

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

Figure 7. Functional architecture for background context monitoring.



is in charge of periodically monitoring the context state and deciding whether possibly occurring context variations demand for the adaptation of the currently viewed page.

In order to be able to take a decision about whether adaptivity actions are to be triggered or not, 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 provides the CM Client with an updated picture of the effective context state. By comparing the state of the (server-side) context model acquired by the current polling with the one acquired by the last polling (or the state at page computation time), the CM Client knows whether the state has changed. If the state has changed, the CM Client asks the Web application for a refresh of the currently viewed page, i.e. the adaptation; if the state has not changed the CM Client proceeds with the monitoring of the context state.

Page Context

In general, the *state* of the context is expressed by the values of all the persistent parameters stored in

the context model and of the volatile parameters sensed at the client or server side. However, an individual page's adaptive behavior is typically influenced by only a subset of the overall context data or, more specifically, by a function expressed over context data. The subset of context data corresponds to a page-specific view over the application's context data, narrowing the focus of the context monitoring activity. This observation leads to the definition of a new concept, i.e. *page context*, which can be leveraged to enhance the efficiency of the context monitoring activity: the *page context* of a page corresponds to a page-specific view over the application's context data, capturing all (and only) those context characteristics that effectively determine the adaptive behavior of the page.

Instead of monitoring the whole state of the application's context data, the definition of a page context for each adaptive page enables the context monitoring activity to focus its observation of the context state to the only page context. This implies, that during hypertext specification each adaptivity rule can be related to a subset of context parameters to be controlled, so that rule conditions do not need to check the state of the whole context model.

Page Context Parameters

In line with the idea of page context, the CM focuses its attention only to the subset of context data in the context model that really determines the adaptive behavior of the viewed page. This implies explicit knowledge of the pages' page context, which can be achieved by defining proper page context parameters for each context-aware page: *page context parameters* define the view over the context model that captures all the static and dynamic properties of a page's page context by means of suitable queries over the context model.

This definition implies that each change to a page context parameter effectively corresponds to a need to adapt the page. The granularity of the *values* of page context parameters must thus be chosen in a way that each change of a parameter value translates into the triggering of the page's adaptivity rule. Each C-labeled page in the adaptive hypertext model is thus associated with an individual page context by means of proper page context parameters stored in the textual representation of the WebML schema, as they are not conveniently expressible in a visual manner. Page context parameters are expressed by means of parametric queries over the context data, where the parameters correspond to client- or server-side context parameters.

Context Digest

In order for the CM to be able to decide whether adaptivity is required, changes to the page context (i.e. the page context parameters) must be communicated from the CM Server to the CM Client. In order to enhance the efficiency of the overall context monitoring activity, the state of the page context is not communicated from the CM Server to the CM Client in form of the set of page context parameters, but instead it suffices to transmit and compare a numeric digest computed over the respective page context parameters, as

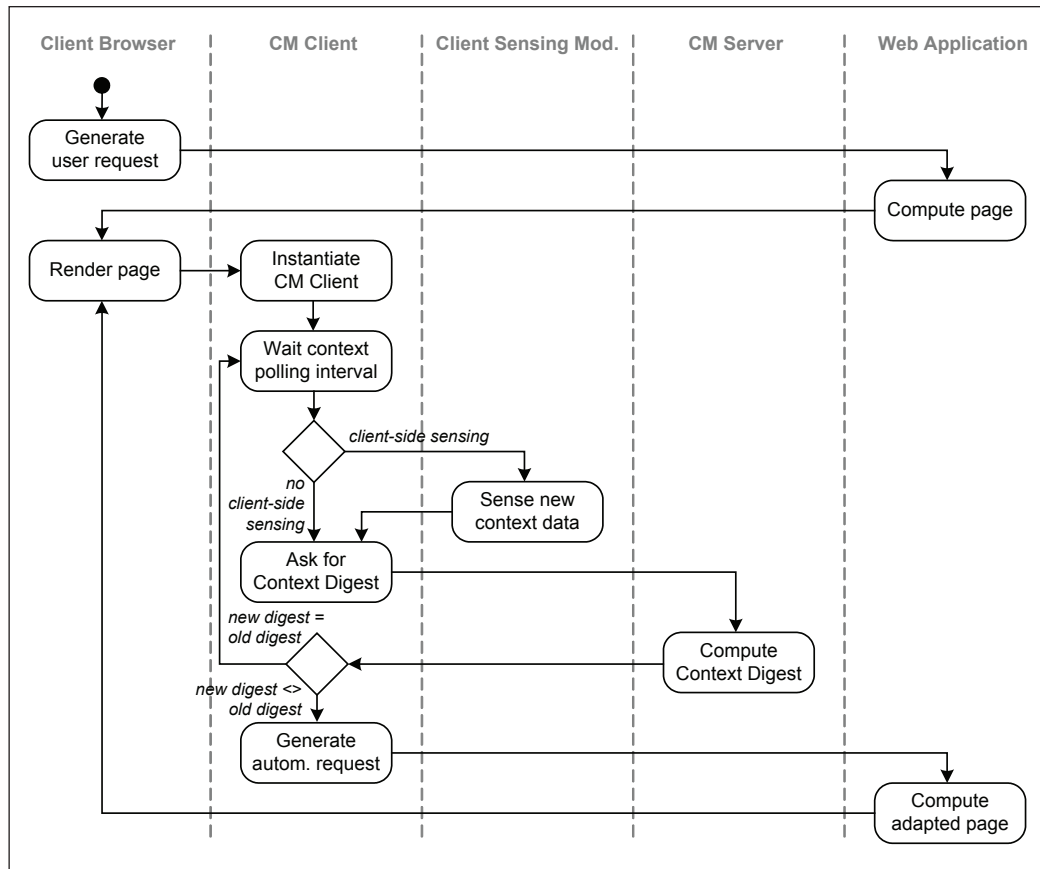
each change to the values of the page context parameters also results in a change of the numeric digest. We call such a numeric digest context digest: the *context digest* corresponding to the page context of a page is the numeric checksum computed over the ordered list of page context parameters.

The context digest is the basis for the decisions to be taken by the CM Client: its values identify variations in the page context, which correspond to the need to adapt the page. The decision is based on the comparison of the current context digest with the last context digest; the first context digest, i.e. when the user accesses the page, is initialized with the context digest valid during page computation.

Figure 8 details the resulting flow of activities enabling the active behavior of the application and shows how the single modules cooperate in order to determine whether adaptivity is required or not. The diagram has one start node (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 an explicit user navigation leading the user to another C-page (which corresponds to starting again from the start node of the diagram and monitoring the Page context of the new page) or to a conventional page (which does not cause any context monitoring activity).

Note that the described mechanism assumes that connectivity is available during the viewing of a C-page in order for the CM client to be able 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 however programmed to manage possible lacks of connectivity and therefore does not generate errors, with the only side effect that adaptivity is suspended until the connectivity is restored.

Figure 8. Background context monitoring for active context-awareness (with client-side context sensing): communicating context data and triggering adaptivity.



APPLICATION IMPLEMENTATION

The extensions that have been introduced into the WebML development method to cope with the new requirements posed by context-awareness and adaptivity in Web applications have been implemented as prototype extension of the WebRatio CASE tool, the official WebML modeling tool, equipped with a powerful automatic code generator. Due to implementation restrictions imposed by the modeling tool, the implementation of the adaptivity logic slightly differs from the models described in this paper (e.g. it was not possible to implement context-aware containers

or to place all the adaptivity operations outside pages). Nevertheless, the described expressive power for the specification of adaptivity rules could be preserved.

Figure 9 shows a screenshot of the WebRatio tool at work. The figure shows the WebML hypertext model of the Buildings page of the PoliTour application, along with its adaptivity logic: two Get ClientParameter units access the GPS coordinates and pass them to the C-label, which forwards them to the outer adaptivity logic (cf. Figure 6). Starting from the shown hypertext model, the PoliTour application has been automatically generated on top of a J2EE platform. The configuration of the

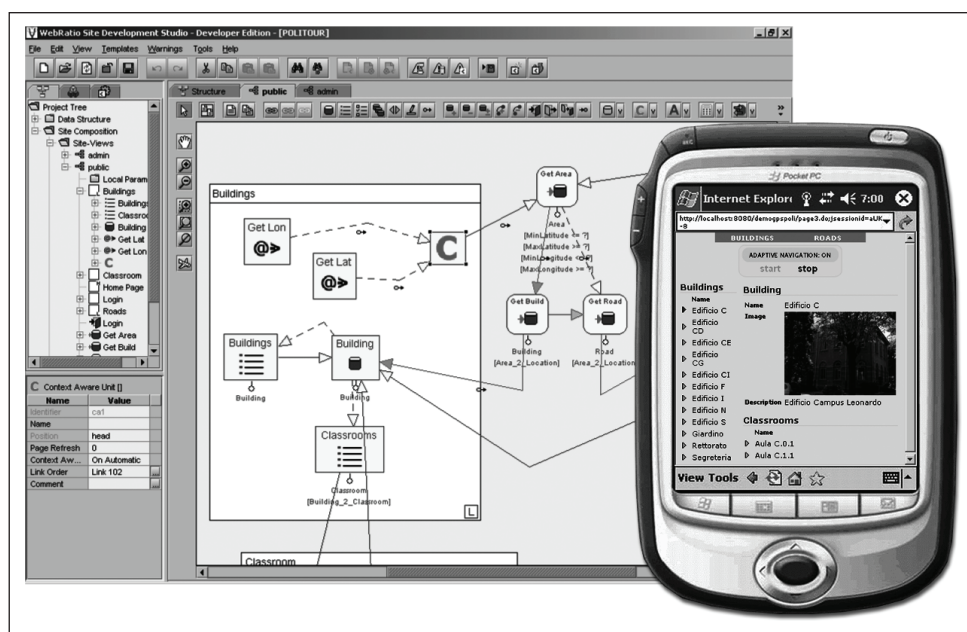
Context Monitor has been performed manually. To access GPS position data, we leverage a client-side Bluetooth GPS device, interfaced via the Chaeron GPS Library (<http://www.chaeron.com/gps.html>) and wrapped by means of Flash (to exchange position data between the CM Client and the GPS library). The WiFi RSSI indicator is acquired in the PDA using Place Lab (<http://www.placelab.org>).

RELATED WORKS

Several other well-established, conceptual design methods have been so far extended to deal with Web application adaptations. Frasinicar & Houben (2002), for example, 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 (De Bra et al., 2003). The UWA Consortium proposes WUML (Kappel et al., 2001) for conceptual hypertext design. Adaptation requirements are expressed by means of OCL-based customization rules, referring to UML class or package elements. Casteleyn et al. (2003) present an extension of WSDM (De Troyer & Leune, 1998) 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 that can be applied to nodes (deleting/adding nodes), information chunks (connecting/disconnecting chunks to/from a node), and links (adding/deleting links). Baumeister et al. (2005) explore Aspect-Oriented

Figure 9. The WebRatio CASE tool showing the hypertext model of the buildings page with respective adaptivity actions and the generated PoliTour application running on a PDA.



Programming techniques to model adaptivity in the context of the UML-based Web engineering method UWE. Recently, WebML (Ceri et al., 2002) has been extended to cover adaptivity and context-awareness (Ceri et al., 2007). 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.

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 (Alferes et al., 2005; Bailey et al., 2002; Bonifati et al., 2002), ECA rules have been 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 (Garrigos et al., 2005a) 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 (Garrigos et al., 2005b), but its real potential for adaptivity management also at runtime remains unexplored.

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. There are however also co-called transcoding solutions, which adopt active rules for adapting Web pages. Most of them focus on the presentation layer and provide mechanisms to transform HTML pages according to (possibly limited) device capabilities (Hori et al., 2000) or users' visual disabilities (Yesilada et al., 2004). Moreover, they typically support only adaptability

and modify Web pages in relation to a static set of user or device parameters. Fiala and Houben (2005) adopt the transcoding paradigm for the development of the Generic Adaptation Component (GAC). GAC provides a broad range of adaptation behaviors, especially supporting runtime 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 any Web site architecture.

CONCLUSION AND FUTURE TRENDS

In this chapter, we have proposed a model-driven approach to the development of context-aware Web applications, an increasingly relevant kind of applications on the Web. We have shown that context-awareness is a first-class design concern that can considerably be aided by model-driven development techniques. But we have also shown that properly dealing with context-awareness and adaptivity at the conceptual level requires extending the expressive power of the adopted conceptual application model, so as to provide developers with suitable modeling constructs and implementation abstractions, proper of such new class of application features. In this chapter, such extensions have been introduced into the already well-established WebML modeling language, but in a similar way we could have also opted for another modeling language, as the ideas and concepts introduced in this chapter are general enough in nature to be applied to other conceptual models as well.

For the future, we believe that a *decoupled runtime management* of adaptivity features will represent a next step in the area of adaptive Web applications. The development of Web applications is more and more based on fast and incre-

mental deployments with multiple development cycles. The same consideration also holds for context-aware and adaptive Web applications and their adaptivity requirements. In (Daniel et al., 2008) we describe our first results obtained with a decoupled environment for the execution and the administration of adaptivity rules. The described approach allows us to abstract 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 line with the current hype of so-called Web 2.0 applications, we are also working on the *mash-up* of context-aware Web applications, in the context of our component-based development method for Web applications called Mixup (Yu et al., 2007). The final goal of the work is to enable even end users to mash up their own context-aware applications, starting from a set of so-called context components and other components equipped with own user interface (which is used to build up the user interface of the mash-up application). Mash-up development is assisted by an easy-to-use and intuitive graphical development environment that supports a drag-and-drop development and by a light-weight runtime environment that is able to interpret and run the mashup, both fully running in the client browser and based on AJAX technology.

REFERENCES

- Alferes, J. J., Amador, R., & May, W. (2005). A General Language for Evolution and Reactivity in the Semantic Web. In *PPSWR'05* (pp. 101–115). Dagstuhl, Germany: Springer.
- Bailey, J., Poulouvassilis, A., & Wood, P. T. (2002). An Event-condition-action Language for XML. In *WWW'02* (pp. 486–495). Honolulu, Hawaii: ACM.
- Baummeister, H., Knapp, A., Koch, N., & Zhang, G. (2005). Modelling Adaptivity with Aspects. In *ICWE'05* (pp. 406–416). Sydney, Australia: Springer.
- Bonifati, A., Braga, D., Campi, A., & Ceri, S. (2002). Active XQuery. In *ICDE'02* (pp. 403–412). San Jose, California: IEEE.
- Brambilla, M., Ceri, S., Comai, S., Fraternali, P., & Manolescu, I. (2003). Specification and Design of Workflow-Driven Hypertexts. *Journal of Web Engineering*, 1(2), 163–182.
- Brusilovsky, P. (1996). Methods and Techniques of Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3), 87–129.
- Casteleyn, S., De Troyer, O., & Brockmans, S. (2003). Design Time Support for Adaptive Behavior in Web Sites. In *SAC'03* (pp. 1222–1228). Melbourne, Florida: ACM.
- Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., & Matera, M. (2002). *Designing Data-Intensive Web Applications*. San Francisco, CA: Morgan Kauffmann.
- Ceri, S., Daniel, F., Matera, M., & Facca, F. M. (2007). Model-driven Development of Context-aware Web Applications. *ACM Transactions on Internet Technologies*, 7(1), article no. 2.
- Daniel, F., Matera, M., & Pozzi, G. (2008). Managing Runtime Adaptivity through Active Rules: the Bellerofonte Framework. *Journal of Web Engineering*, 7(3), 179–199.
- De Bra, P., Aerts, A. T. M., Berden, B., de Lange, B., Rousseau, B., Santic, T., Smits, D., & Stash, N. (2003). AHA! The Adaptive Hypermedia Architecture. In *Hypertext'03* (pp. 81–84). Nottingham, UK: ACM.
- De Troyer, O., & Leune, C. J. (1998). WSDM: A User Centered Design Method for Web Sites. *Computer Networks*, 30(1-7), 85–94.

- Dey, A. K., & Abowd, G.D. (2000). Towards a Better Understanding of Context and Context-Awareness. In *CHI'00 Workshop Proceedings*, The Hague, The Netherlands.
- Fiala, Z., & Houben, G.-J. (2005). A generic transcoding tool for making web applications adaptive. In *CAiSE'05 Short Paper Proceedings*, volume 161 of CEUR Workshop Proceedings. CEUR-WS.org.
- Frasincar, F., & Houben, G.-J. (2002). Hypermedia Presentation Adaptation on the Semantic Web. In *AH'02* (pp. 133-142). Málaga, Spain: Springer.
- Garrigós, I., Casteleyn, S., & Gómez, J. (2005a). A Structured Approach to Personalize Websites Using the OO-H Personalization Framework. In *APWeb'05* (pp. 695-706). Shanghai, China: Springer.
- Garrigós, I., Gómez, J., Barna, P., & Houben, G.-J. (2005b). A Reusable Personalization Model in Web Application Design. In *WISM'05* (pp. 40-49). Sydney, Australia.
- Henricksen, K., & Indulska, J. (2004). Modelling and Using Imperfect Context Information. In *PERCOMW'04* (pp. 33-37). Washington, United States: IEEE.
- Henricksen, K., Indulska, J., & Rakotonirainy, A. (2002). Modeling Context Information in Pervasive Computing Systems. In *Pervasive'02* (pp. 167-180). London, UK: Springer.
- Hori, M., Kondoh, G., Ono, K., Hirose, S., & Singhal, S. K. (2000). Annotation based Web Content Transcoding. *Computer Networks*, 33(1-6), 197-211.
- Kappel, G., Pröll, B., Retschitzegger, W., & Schwinger, W. (2001). Modelling Ubiquitous Web Applications - TheWUML Approach. In *ER'01 Workshops* (pp. 183-197). Yokohama, Japan: Springer.
- Lei, H., Sow, D. M., Davis, J. S. II, Banavar, G., & Ebling, M. R. (2002). The design and applications of a context service. *SIGMOBILE Mobile Computing and Communications Review*, 6(4), 45-55.
- Schilit, B.N., & Theimer, M.M. (1994). Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5), 22-32.
- Web Models s.r.l. (2008). WebRatio Site Development Studio. Retrieved January, 2008, from <http://www.webratio.com>.
- Yesilada, Y., Harper, S., Goble, C. A., & Stevens, R. (2004). Screen readers cannot see: Ontology based semantic annotation for visually impaired web travellers. In *ICWE'04* (pp. 445-458). Munich, Germany: Springer.
- Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., & Matera, M. (2007). A Framework for Rapid Integration of Presentation Components. In *WWW'07* (pp. 923-932). Banff, Canada: ACM.