Model-driven Development of Context-Aware Web Applications

Stefano Ceri, Florian Daniel, Maristella Matera, Federico M. Facca
Dipartimento di Elettronica - Politecnico di Milano
P.zza Leonardo da Vinci, 32 - 20133 - Milano, Italy
{ceri,daniel,matera,facca}@elet.polimi.it

Context-aware, multi-channel Web applications are more and more gaining consensus among both content providers and consumers, but very few proposals exist for their conceptual modeling. This paper illustrates a conceptual framework that provides modeling facilities for context-aware, multi-channel Web applications; it also shows how high-level modeling constructs can drive the application development process through automatic code generation. Our work stresses the importance of user-independent, *context-triggered* adaptation actions, in which the context plays the role of a "first class" *actor*, operating independently from users on the same hypertext the users navigate. Modeling concepts are based on WebML (Web Modeling Language), an already established conceptual model for data-intensive Web applications, which is also accompanied by a development method and a CASE tool. However, given their general validity, the concepts of this paper shape up a complete framework that can be adopted independently from the chosen model, method, and tool.

Categories and Subject Descriptors: H.1 [**Information Systems**]: Models and Principles; H.5.4 [**Information Interfaces and Presentation (e.g., HCI)**]: Hypertext/Hypermedia; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*

General Terms: Design, Languages

Additional Key Words and Phrases: Context, Context-awareness, Context-aware Web Applications, Conceptual Modeling, WebML, Adaptive Hypertext, Adaptive Hypermedia

1. INTRODUCTION

As Web applications spread in almost every domain, novel challenges are posed to developers. The current advances in the communication and network technologies are changing the way people interact with Web applications, providing them with different types of mobile devices for accessing at any time from anywhere and with any media services and contents customized to users' preferences and usage environments. More and more users themselves ask for services and applications highly tailored to their special 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. Due to such premises, new issues

Author's address: Florian Daniel, Dipartimento di Elettronica e Informazione, Politecnico di Milano, P.zza L. da Vinci, 32 - 20133 - Milano - Italy.

and requirements need to be addressed for supporting *context-aware* and *multi-channel* access to services and applications.

Context-awareness is often seen as recently emerged research field within information technology, and in particular within the domain of the Web. From another perspective, it can be however interpreted as natural evolution of personalization, addressing not only the user's identity and preferences, but also the 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 efficiency of the overall communication of contents. Context-awareness goes one step further in the same direction, aiming at enhancing the application's usefulness by taking into account a wider range of properties than personalization.

The second ingredient for modern Web applications, *multi-channel access*, is gaining as well increasing consensus among both content consumers and providers. While the former are more and more attracted to portable devices equipped with high-resolution color displays, able to provide similar browsing experiences as traditional desktop computers, the latter are increasingly facilitated by standardized communication protocols (i.e., HTTP) and markup languages (i.e., HTML), supported by most of such devices. Consequently, multi-channel deployment does not anymore require completely different, parallel design approaches and is thus becoming rather a presentation problem on top of unified engineering solutions. Cost-effective multi-channel development is therefore becoming possible, thus finally raising the inclination of content providers toward it.

In this paper we try to combine the potential behind the previous considerations into a conceptual framework, providing modeling facilities for context-aware, multi-channel Web applications. Conceptual modeling methods have already proved their effectiveness for the design of personalized Web applications (see for example [Ceri et al. 1999; Schwabe et al. 2002]). However, very few proposals exist for the conceptual modeling of reactive, context-aware Web applications. This paper will therefore introduce some modeling primitives able to capture the semantics of reactive behaviors, and will also show how high-level constructs can drive the development process through automatic code generation.

Differently from most conventional adaptive hypermedia systems, which mainly address the problem of adapting the results of user-generated requests, our work also stresses the importance of user-independent, *context-triggered* adaptation actions, which finally leads to interpret context as a "first class" *actor* operating independently from users on the same hypertext the users navigate.

Modeling concepts at the basis of the proposed approach, as well as our implementation experiences will be introduced in the context of WebML (Web Modeling Language), an already established conceptual model for data-intensive Web applications, which is also accompanied by a development method and a CASE tool [Ceri et al. 2002a; Ceri et al. 2002b; Ceri et al. 2003]. However, given their general valence, the introduced concepts suggest a design methodology that can be also adopted independently from the chosen modeling notation and method.

This paper is organized as follows. Section 2 recalls some basic concepts about context-awareness and reviews some notable works proposed in literature, with the aim of highlighting the rationale and the background of our research. Section 3

investigates the main requirements that emerge when Web applications need to be augmented through context-aware mechanisms. Section 4 introduces a model-based conceptual view over context-awareness for Web applications, and translates the observed requirements into proper modeling concepts. As we adopt WebML as example modeling language for expressing our vision on context-aware applications, Section 4 also provides a comprehensive summary of this model. Section 5 then illustrates a modeling example taken from some experiences gained within an Italian national research project (MAIS). Section 6 shows how possible conflicts among concurrent actions by users and context, operating over the same hypertext, can be resolved. Next, Section 7 gives an overview over our current implementation. Finally, in Section 8 we draw some conclusions and also discuss our future work.

2. RATIONALE AND BACKGROUND

Although many definitions of context are given by enumerating examples or by choosing synonyms[1], inspired by the work of Dey and Abowd [2000], 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.*

Context can be therefore used for achieving more effective and efficient interactions in all those situations where the contents and services offered by the application strongly depend on the current environmental situations, users' (dis)abilities, and/or the actual purpose of the application. Several situations demanding for adaptivity might arise:

—**Adaptive personalization.** User profile attributes for personalization purposes may present different levels of variability in time, which may comprise (fast) changing properties (e.g. pulse frequency) as well as static ones, such as the name of the respective user. Adaptive personalization mechanisms that take into account such profile peculiarities allow going beyond common, rather static content tailoring.

—**Functional needs.** Applications may depend intrinsically and in a structural manner from context data. Location-aware applications, city map services or navigation systems, for instance, treat position information as core content, and proper localization mechanisms must be supported. For such kind of applications, the use of context represents a mandatory functional requirement, rather than an optional feature.

—**Exception handling.** Adaptive or context-aware mechanisms are particularly suited for handling exceptional situations with respect to expected application

[1]Rather than defining context by means of synonyms as well as for better readability, throughout this paper we will use the terms *adaptive*, *reactive*, or *context-sensitive system* when speaking about context-aware applications. Also, we will speak about *customization* or *personalization* with respect to the user's identity, and about *adaptation* or *adaptivity* when referring to context-driven adjustments or modifications.

behaviors. Critical events may raise exceptions and require proper reactions being performed. Workflow-driven hypertexts, for example, represent a class of applications that could benefit from adaptive management features for coping with exceptions. Also, alerting mechanisms as required for critical production processes or pro-active maintenance approaches could be achieved.

—**Interaction-enabling functionalities.** Context could as well consider handicaps or physical disabilities of users, such as vision problems, blindness or paralysis, for adapting the application accordingly and providing alternative, suitable interaction mechanisms and modalities. In that sense, adaptivity can provide functionalities enabling handicapped users to properly interact with applications.

—**Enhanced effectiveness.** Several other context parameters can be exploited for providing appropriate contents and program features at the right time, priority and emphasis. For example, specifically targeted special offers can be advertised and directed more efficiently or, also, presentation properties can be adapted to varying luminosity conditions for better readability. The overall effectiveness of applications can thus be significantly enhanced by means of adaptive or context-aware extensions.

Based on the previous considerations, context can be described in terms of properties and attributes related to the current user, her/his current activities, the location in which the application is used, the devices, and some other aspects of the environment and of the application itself that can be used for determining the required adaptation in certain situations [Kappel et al. 2003; Kobsa et al. 2001]. Also, for being able to manifest active or reactive behaviors, context-awareness further requires automatisms on the application part that may be triggered by changes of anyone of the parameters that make up context.

2.1 Related Work

So far context-awareness has been mainly studied in the fields of ubiquitous, wearable or mobile computing. A significant number of applications have been successfully developed [Want et al. 1992; Long et al. 1996], and context abstraction efforts have produced proper platforms or frameworks for rapid prototyping and implementation of context-aware software solutions [Salber et al. 1999].

Within the domain of the Web, so-called *adaptive hypermedia* systems [Brusilovsky 1996] address advanced adaptation and personalization mechanisms, and recent research efforts also address the special needs of mobile Web applications and portable device characteristics. *HyCon* [Hansen et al. 2004], for example, represents a general platform for the development of context-aware hypermedia systems with special emphasis on location-based services. In addition to proper location-sensing devices (like GPS receivers), support for other local and remote context sensing devices is provided. Example *HyCon* applications range from location-based browsing and annotation to geo-based search support, and essentially make use of GPS coordinates. The main drawback of the approach proposed by the authors, however, lies in the fact that a proprietary Web browser (*HyConExplorer*) is required.

The *AHA!* system proposed by De Bra et al. [2003], on the other hand, builds on top of standard Web technologies and 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). AHA! is delivered as Open Source software and provides a versatile adaptive hypermedia platform for the development of online courses, museum sites, encyclopedia, etc.

Belotti et al. [2004] address the problem of fast and easily developing context-aware (Web) applications along a technological, database-driven approach, based 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 [Grossniklaus and Norrie 2002]. In [Belotti et al. 2004] the authors 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. Also, they discuss the twofold role the content management system can play within a context-aware application, namely it can act as both consumer and provider of context information.

On the other side, some well known model-driven methodologies, such as *HDM* [Garzotto et al. 1993], *OOHDM* [Schwabe et al. 1996], *RMM* [Isakowitz et al. 1995], *UWE* [Koch et al. 2001], and *Hera* [Vdovjak et al. 2003] aid developers in the design of Web information systems, providing modeling primitives for shaping the application structure and behavior at a conceptual level. Despite the numerous advantages offered by conceptual, model-based development of Web applications [Fraternali 1999], only few attempts exists that aim at modeling also adaptive or context-aware behaviors at a conceptual level. For instance, AHA! is inspired by *AHAM* [De Bra et al. 1999], a Dexter-based reference model for adaptive Web applications with a heavily educational background. According to AHAM, adaptive hypermedia applications are based on a *domain model*, a *user model* and a so-called *teaching model*. Adaptation is achieved at runtime by adapting contents from the domain model according to *rules* of the teaching model taking into account properties of the user model. Unfortunately, the teaching-learning paradigm restricts the AHAM applicability to specific domains, and also makes the method less suitable for modeling generic adaptive hypermedia systems. Also, AHAM's adaptivity mechanisms work at instance level, and are thus not very suitable for coping with the huge amount of (dynamic) contents that today's Web applications may require.

Within the *AMACONT* project, on top of the model-based framework of the *Hera* project, Fiala et al. [2004] propose a component-based XML document format for the implementation and deployment of component-based, adaptive Web presentations. 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 schemes. However, the Hera methodology adopted for specifying the application schemas is based on a conventional, non-adaptive interpretation of hypertext.

Barna et al. [2004] address exactly the above mentioned lack of dynamism at model level, and show how the *Hera* design methodology [Vdovjak et al. 2003] can be used successfully for the design of adaptive, dynamic Web applications. According to the authors, through appropriately updating the user model at runtime,

Hera schema views defined over the application data can capture adaptive characteristics. Such schema-level adaptation specifications can (partially) overcome the lack of proper conditional mechanisms for content inclusion/exclusion in the Hera methodology. Like in AHAM, adaptation is mainly based on properties of the user model, and customizations according to a broader interpretation of *context* are not tackled.

The modeling approach presented in this paper capitalizes over previous works, trying to overcome their deficiencies. Our method provides concepts, notations and implementation technologies for the development of Web applications, without posing any constraint over their target domain, neither requiring proprietary client- or server-side solutions. It takes advantage of the model-based paradigm, which offers some high-level intuitive constructs for specifying the application structure and behavior. Due to the formal definition of construct semantics, the resulting specifications can then be transformed into running code, by means of consolidated generative techniques [Ceri et al. 2003], thus facilitating the overall development process. As better explained in the following sections, our method also relies on a data-driven paradigm, which fosters the representation of personalization and context meta-data within the application data source, thus leading to Web applications where content customization and context-based reactive behavior are achieved in a totally dynamic wise. Additionally, the main novelty with respect to other model-based approaches is the promotion of context as "first class" actor, which leads to reactive applications, able to respond autonomously to detected context changes.

3. CHARACTERIZING CONTEXT-AWARE APPLICATIONS

In line with most conceptual modeling approaches for Web application design [Fraternali 1999], the method we propose in this paper builds on a strong separation of concerns among *data* and *hypertext* design. The former activity aims at representing the organization of the application data source; the latter supports the specification of the application front-end, i.e., of content organization within pages and of the invocation of server-side operations implementing the application's business logic. Also, our modeling approach is data-driven, since the overall design process starts from data design, and most decisions during the successive phases strongly depend on data organization.

With respect to the modeling of conventional Web applications, the introduction of context as an actor, further determining the behavior of a Web application, poses new requirements. We will thus concentrate on the effect of such requirements over the two above design dimensions, and in particular on *(i)* the specification and management of context data, and *(ii)* on the hypertext augmentation with reactive behaviors as response to context state changes.

3.1 Functional System Requirements

According to the data-driven paradigm, the use of context within applications primarily requires its representation at data level. Also, mechanisms for context data acquisition and access are needed [Kobsa et al. 2001]. Therefore, the following issues become relevant for managing data sources in context-aware applications:

(1) *Context model definition and representation* within the application data source. The main context properties needed for supporting adaptivity must indeed 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.
   (b) *Context model updating*, for keeping context data consistent and up-to-date with respect to the actual environmental conditions.
   (c) *Context data monitoring*, for detecting those variations in context that trigger adaptivity actions. Any variation may therefore cause an automatic (*context-triggered*) adaptive behavior of the Web application.

Given the previous characterization of context data and of mechanisms for context management, some other funtional requirements are concerned with adaptivity actions over the application hypertext. According to our approach, context-aware applications must therefore ensure:

(1) *Adaptivity of contents and services delivered by accessed pages*, on the basis of the current context.

(2) *Adaptivity of navigation*, through automatic navigations toward other pages, which are more appropriate for the reached context.

(3) *Adaptivity of the whole hypertext structure*, for facing coarse-grained adaptation requirements, for example due to changes of user's device, role and activity within a multi-channel, mobile environment.

(4) *Adaptivity of presentation properties*, in order to allow for more fine-grained adjustments of the application's appearance.

3.2 Architectural Requirements

In order to achieve the previous goals, it is necessary to monitor some context data, and capture them from the environment in which the application is executed. Figure 1 shows the context data flow within a possible architecture tailored to support context-aware hypertext solutions. The data source includes both the application data (i.e., the business objects that characterize the application domain), and some context data, the latter offering at any moment an updated representation of the context status, which we call *context model*. At runtime, the dynamic computation of the hypertext primarily exploits the application data for populating pages and units. We then assume that a subset of hypertext elements, included in the so-called *adaptive hypertext*, be augmented with adaptive behaviors, thus their computation exploits also context data.

As expressed in Figure 1, the proposed architecture takes into account context-sensing mechanisms for capturing context data from the application execution environment. This may imply using standard (off-the-shelf) sensing components. It may also require developing proprietary sensing mechanisms on the client side, as it happens for GPS-equipped devices, or adopting centralized sensing infrastructures collecting context data in the usage environment and communicating them back to the application. Examples of centralized sensing are given by the *Active Badge*

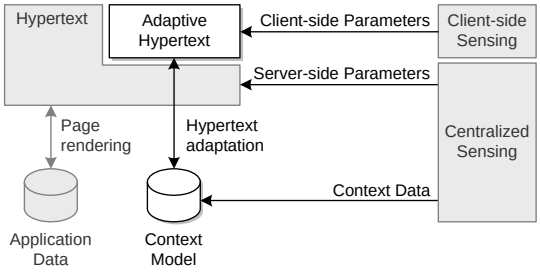

Fig. 1. Context Data Flow.

location system [Want et al. 1992], based on infrared signals, or the *LANDMARC* system [Ni et al. 2004], based and RFID (Radio Frequency Identification).

However, from a conceptual point of view, the adopted context sensing mechanisms do not affect much the specification of the application hypertext: the only relevant aspect is the way sensed data are communicated back to, and managed by the application. Therefore, our modeling method assumes that some solutions allow sensing such data, that become available to the application in three modalities:

(1) Some context parameters generated at client-side are sent back to the application, for example as values appended to the URL of requested pages or by means of SOAP messages.
(2) The values of some HTTP session parameters, managed at server-side, are set according to newly captured context data.
(3) The sensing mechanism updates the context model at data level, for example through asynchronous services.

The first two acquisition mechanisms act at the hypertext level, and do not operate directly on the data layer. However, some operations in the adaptive hypertext must provide support for storing the fresh captured values, thus keeping the context model at data level updated with respect to the observed context state. The last acquisition mechanism, which operates on data, is particularly suited for those adaptive systems that build on centralized context sensing mechanisms, such as RFID.

4. MODELING CONTEXT-AWARE WEB APPLICATIONS

The previously outlined requirements can be translated into modeling concepts and primitives, able to express at a high level the organization of context data and the augmentation of the application through adaptive behaviors. This section therefore introduces some high-level abstractions that are required to support context awareness. It also shows how such abstractions can be captured and made concrete within a specific conceptual model, WebML, through the definition of some corresponding modeling primitives extending the set of constructs already offered for "conventional" Web applications.

In order to facilitate the comprehension of the introduced concepts, this section starts by shortly recalling WebML. The reader already familiar with this model is referred to the remaining sections.

4.1 WebML: an Overview

WebML is a visual language for specifying the content structure of Web applications and the organization and presentation of contents in one or more hypertexts. [Ceri et al. 2002a; Ceri et al. 2002b].

The design process 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.

The *WebML Hypertext Model* then 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* to constitute site views. Links can connect units in a variety of configurations, yielding to composite navigation mechanisms. Besides representing user navigation, links between units also specify the transportation of some parameters that the destination unit uses 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 through the `connect` and `disconnect` units).

WebML also provides units for the definition of *session parameters*. Parameters can be set through the `Set unit`, and consumed within a page through a `Get unit`.

Figure 2 depicts an example WebML hypertext model. It shows a site view with three pages, which allow users to see their reviews about a museum's artworks, previously submitted through the Web application, and to modify them. In the `Home` page a `Get` unit reads the identifier of the current user from the session parameter `CurrentUser` and provides it in input to the following `Data` unit, which thus publishes the user's personal profile data. The data to be published are restricted by means of a selector condition, specified below the unit. The user identifier is further propagated to the `Index` unit `My Reviews` by means of a so-called *transport link*. In the `Review` page, a `Data` unit shows a review previously selected through the home page's `My Reviews` index. Also, a link leads the user to the `Modify Review`

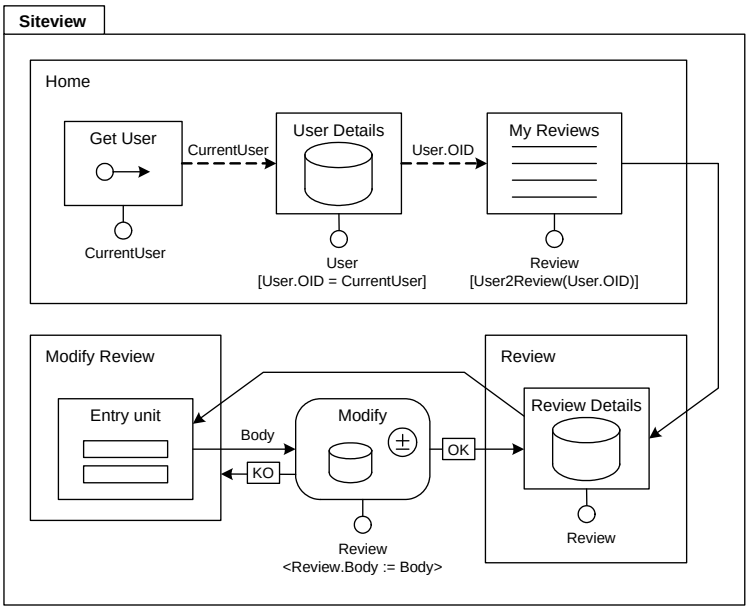

Fig. 2. An example of WebML hypertext schema.

page, which includes an `Entry` unit for inserting new text for the modification of the review. The form submit link than activates a `Modify` operation, which updates the review data within the application data.

Besides having a visual representation, WebML primitives are provided with an XML-based textual representation, which specifies additional detailed properties, not conveniently expressible in the graphic notation. Web application design based on WebML can be therefore represented as visual diagrams, as well as XML documents. The availability of an XML specification also enables the deployment of the same design into multiple rendering formats, such as HTML (which is the standard choice for deployment), but also WML [Hjelm et al. 1998], SALT [SALTforum.org 2005] or VoiceXML [W3C 2004] for multi-modal interactions. This feature greatly supports the development of multi-channel applications.

For a more detailed and formal definition of WebML, the reader is referred to [Ceri et al. 2002a].

4.2 Modeling Context as Data

In accordance with our data-centric approach, we now enrich the application data source with a *context model* to keep a consistent and updated representation of meta-data needed for supporting adaptivity. As better shown later on in this paper, by explicitly representing context properties within the data source, many useful customization policies can be expressed declaratively in the hypertext specification, instead of being buried in the source code of the application.

Context data can derive from several sources integrating sensed, user-supplied and derived information [Henricksen and Indulska 2004; Henricksen et al. 2002]. While user-supplied data are generally reliable and tend to be static, sensed data

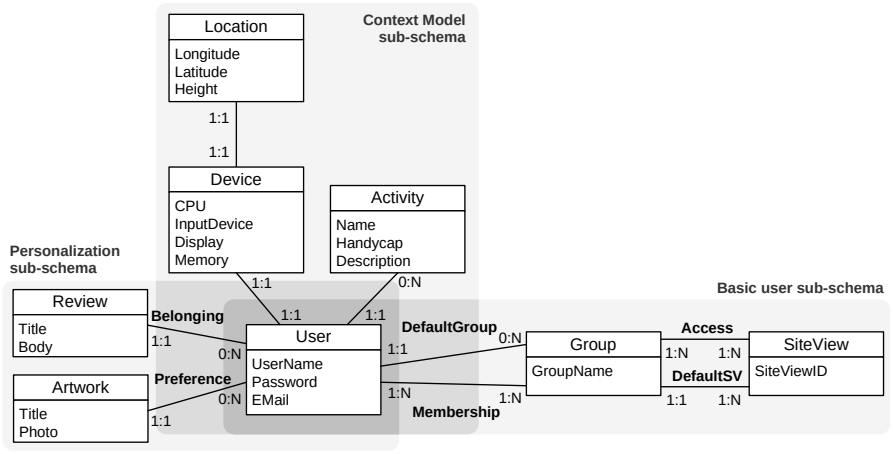

Fig. 3. Combined data design of user, personalization, and context sub-schemas. The entity `User` is the center of the three sub-schemas, providing the starting point for navigating context data.

are highly dynamic and can be unreliable due to noise and sensor errors. The problem of unreliability has been already addressed by some modeling solutions, which associate context information with quality data [Lei et al. 2002]. Our approach does not explicitly cover these issues, although we recognize the distinction between physical and logical context data, the latter being a transformation of the former providing meaningful abstractions with respect to the application domain [Schmidt et al. 1999]. Such transformation can be achieved by means of proper filters over raw context data. Our modeling approach therefore assumes that data feeding the context model are normalized with respect to such filtering.

Also, we assume that the actual context model of a real application can vary depending on the application domain and, also, on the adaptivity goals to be fulfilled. 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, we do not prescribe a precise, rigid characterization of context; rather we introduce some guidelines on how to extend an application's data source with context meta-data.

Figure 3 illustrates an Entity-Relationship diagram exemplifying an application data source with a possible context model. As can be noted, we propose augmenting the application data schema with three sub-schemas that serve the following purposes:

—**User profile sub-schema.** Users, groups and site views are represented as "first-class citizens" in the application data source. The entity `User` provides a basic profile of the application's users, the entity `Group` allows managing access rights, and the entity `Site View` allows representing and associating views over the application's data source tailored to the needs of the respective user group. The many-to-many relationship `Membership` expresses that users may belong to multiple groups, which in turn cluster multiple users. The relationship `Default-`

`Group` connects a user to his/her default role and, when logging into the application, by means of the relationship `DefaultSV` the user can be forwarded to the default site view of his/her default group. The many-to-many relationship `Access` describes which site views a specific group can access. This relationship is required only in the case of adaptive applications that may require different interaction and navigation structures for a same group, according to varying context properties. Therefore, depending on the context state, the application is able to determine and forwarding the user to the most appropriate site view.

—**Personalization sub-schema.** Figure 3 also shows a possible personalization sub-schema mainly consisting of some relationships connecting entities of the application schema to the entity `User`. For instance, the entity `Review` is connected to the entity `User` by means of the relationship `Belonging`, which expresses to which user reviews posted to the Web application belong. Similarly, the relationship `Preference` links a user to the artworks he/she likes most (for example, among those published by a possible museum Web site).
In general, personalization relationships between the entity `User` and some other entities have the meaning that the user is the creator/owner of the specific object, or that he/she has expressed explicit or implicit preferences over it. Similar relationships allow thus personalizing contents with respect to the identity of users. Personalization schemas can vary in complexity, according to the amount of contents to be tailored to individual users.

—**Context model sub-schema.** Finally, Figure 3 proposes a possible configuration of context meta-data, as it could apply, for example, to mobile and multichannel Web applications. The entities `Device`, `Location` and `Activity` describe the particular properties of context considered by the application. As within the personalization sub-schema, context entities are connected to the entity `User` for associating each user to his/her (personal) context.
Applications may require different sets of context entities, according to their functional requirements and goals. Therefore, our approach just prescribes to have such entities associated, directly or indirectly (as in the case of the entity `Location` in Figure 3), to the user, which is the actual starting point for navigating the context model and extracting context information.

In a certain sense, also the relationship `Access` within the user profile sub-schema can be considered being part of the context sub-schema, as it is only justified when context conditions may require the overall hypertext structure to vary according to specific context states. Users belonging to a given group might in fact act in totally different contexts, and might hence require different site views to switch among, as soon as the context changes. The modeling of conventional (i.e., non context-aware) applications, would instead simply require only one relationship associating one user group to one (default) site view.

4.3 Structuring the Context-Aware Hypertext

Besides augmenting the application data schema with context data, our modeling approach also introduces some new constructs at hypertext level, required for specifying adaptive application behaviors. We further clarify how different adaptivity policies, based on automatic polling, can enact the evaluation of adaptivity actions.

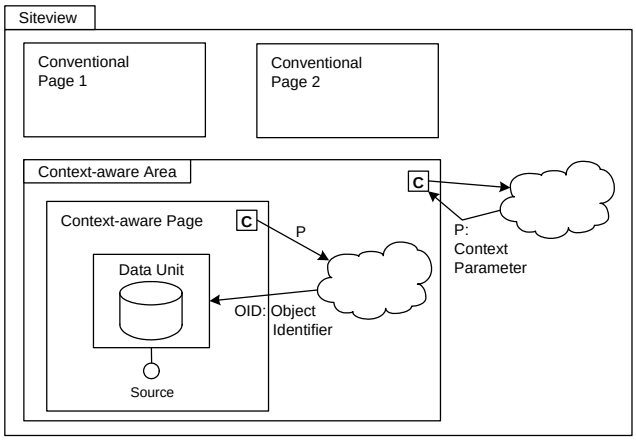

Fig. 4. WebML hypertext schema with two conventional pages, one context-aware page, one context-aware area, together with their context clouds. The parameter `P` exemplifies the propagation of reusable context data through the hierarchical passing of context parameters from an outer area to an inner page.

4.3.1 *Context-aware Pages.* Figure 4 graphically illustrates our vision on context-aware hypertexts. Our basic assumption 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 typical "access pages" (including links to the main application areas) might not be affected by the context of use.

As can be seen in Figure 4, we tag adaptive pages with a `C`-label (standing for *context-aware*) for distinguishing them from conventional pages. This label indicates that some adaptivity actions are associated with the page, and that during the execution of the application such actions must be evaluated prior to the page computation, because they can serve for customizing the page content or modifying the predefined navigation flow.

In order to show a context-aware behavior, each `C`-page must be provided with the capability of monitoring the context state and, based on that, triggering its adaptivity actions. This can be achieved in part by evaluating the adaptivity actions every time the page is accessed, before the actual computation of the page. A different policy may require evaluating and executing the adaptivity actions after the user's access to the `C`-page. When accessed, the page is therefore rendered according to the user selection, and its content is successively adapted by refreshing the page, based on the current context state. The standard HTTP protocol underlying most of today's Web applications implements a strict *pull* paradigm, in which refresh can only occur as response to client-side generated page requests. Therefore, in absence of dedicated server-side *push* mechanisms for delivering updated pages when needed, the HTML `http-equiv META`-option, or also JavaScript, JavaApplets, or Flash scripts, provide valuable client-side mechanisms for "simulating" the required active behavior. More precisely, this simulation implies generating proper HTTP requests toward the application server, which serve a twofold purpose:

—On one hand they provide the necessary polling mechanism for querying the context model and triggering the adaptivity actions attached to the page, thus reacting to possible context changes.
—On the other hand, when the page request carries client-side sensed data, the polling also enables the communication of such context data to the application server.

4.3.2 *Adaptivity Policies.* Based on the previous observations, the evaluation of adaptivity actions for C-pages can be triggered according to two different policies:

—*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, periodic polling generates automatic page requests that trigger the evaluation of the adaptivity actions.
—*Immediate Adaptivity*: context is granted the highest priority. Therefore, the adaptivity actions are evaluated every time the page is accessed, prior to the actual page computation. This means that the adaptivity actions are evaluated at the first time the page is accessed by users, as well as at each automatic polling.

In our approach, we assume the *deferred* adaptivity as default policy: adaptivity is started only by automatic refreshes coming after the user has entered the page. When a user navigates to a particular page, the first generated response always produces the expected results based on user selection; only afterward that page might become subject to adaptation, according to the current context. 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.

The choice of the policy, as well as the specification of the required polling interval must be expressed as properties of C-pages. Therefore, each C-page is associated with the pair of properties $< Adaptivity\_Policy, Polling\_Interval >$. It is worth noting that different C-pages may adopt different adaptivity policies as well as different polling intervals.

4.3.3 *Context Clouds.* We call the set of adaptivity actions attached to a page *context cloud*. As sketched in Figure 4, the cloud is external to the page and the chain of adaptivity actions it clusters is kept separate with respect to the page specification. The aim is to highlight the two different logics deriving from the role played by pages and context clouds: while the former act as providers of contents and services, the latter act as modifiers of such contents and services.

The context cloud is associated to the 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 cloud logic, since it can transport parameters deriving from page contents, which may be useful for computing actions specified in the cloud. Also, on the other way around, a link from the cloud to the page can transport context parameters or in general values computed by the adaptation actions, which might affect the adaptivity of page contents with respect to the renovated context.

Context cloud are computed before the actual rendering of the page, according to the adaptivity policy associated to the page (immediate or deferred).

4.3.4 *Context-Aware Containers.* The central context-aware element is the page. However, as represented in Figure 4, we also propose to define context-aware *containers* (site views and areas, in terms of WebML) as grouping facilities that allow one to insulate, and to specify only once, redundant adaptivity actions to be performed for every C-page within the container. There are in fact actions to be evaluated for every C-page. Specifying such actions once, associated to a page container, allows us to keep the specification easy to read.

Figure 4 also illustrates the possibility of hierarchically passing parameters from an outer cloud to an inner one. More precisely, if the evaluation of an outer cloud produces results to be reused at an inner level, as it happens for some context parameters, it passes such values back to the C-label that activated the computation of the cloud. At the inner level, such parameters can then be "consumed" within context clouds. Parameter passing from outer containers to the current context cloud occurs through the cloud-activating link. At the end of the adaptivity action chain, links exiting from the last evaluated cloud might carry parameter values for the computation of page units.

Typical actions to be specified at the container level are the acquisition of fresh context data and the consequent updating of the context model. We therefore propose two levels for the specification of context-adaptivity actions:

—*Actions for context model management*, addressing operations for context data acquisition and the consequent context model updating. These adaptivity actions need to be executed prior to the execution of any other action within the context cloud, for gathering an updated picture of the current context. Therefore, they can be associated with the most external containers (site views or areas), and are inherited by all the internal containers (areas or pages).

—*Actions for hypertext adaptivity*, addressing the definition of rules for page and navigation customization associated with C-pages.

As for C-pages, each container has its own adaptivity policy (immediate or deferred). When a C-page is requested, the context clouds of its containers (if any), from the outermost container to the innermost, are evaluated immediately (at the first user access) or in a deferred manner (at the page refresh), depending on the value of the `Adaptivity_Policy` property declared for the containers. In general, the container policy is independent from the policy of inner containers and pages[2]. Therefore, it may happen that the actions in a container's context cloud are evaluated immediately, even when the actions associated to the inner containers and pages require a deferred evaluation or vice-versa. In this sense, the context cloud hierarchy also facilitates the specification of different "layers" of adaptivity actions requiring different evaluation policies.

Differently from C-pages, containers do not require the specification of a polling interval, which is derived by the interval associated to the inner C-pages.

[2]The only exception occurs when a dependency exists among clouds at different levels, due to parameter passing.

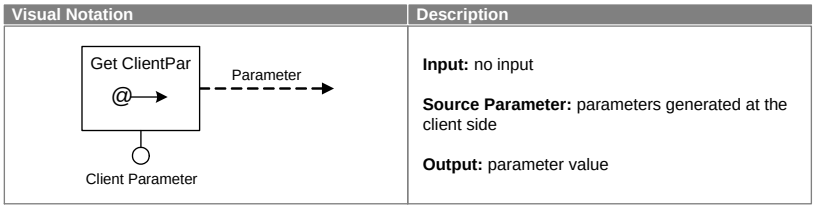

Fig. 5. Visual notation for the *Get URL Parameter* unit.

4.4 Specifying Adaptivity

The main novelties for modeling context-aware pages lay in the specification of actions clustered within context clouds. In the following, we will introduce new WebML modeling constructs that ensure full coverage for the specification of context model management and hypertext adaptivity actions. The latter allow specifying in a visual manner actions for acquiring and updating context data, as well as ECA (Event-Condition-Action) rules, where the *event* consists in the page request, the *condition* for rule activation (if any) consists in the evaluation of context parameters previously acquired, and the *action* consists in adaptations of the hypertext front-end.

4.4.1 *Managing Context Data.* In order to gather adaptivity with respect to the current state of context, the application must be able to acquire and manage context parameters, according to the mechanisms illustrated in Section 3.1. The modeling of conventional applications already provides primitives for managing server-side parameters. For example, in WebML context data made available as HTTP session parameters can be handled by `Get` units (see Section 4.1). However, new primitives and mechanisms are required for:

—*Specifying the acquisition of fresh context data, sensed at the client side.* A new `Get ClientPar` unit (see Figure 5) is needed for supporting the retrieval of parameters generated at the client side and communicated back to the application.

—*Specifying the acquisition of context data from the context model.* In conventional applications, page computation implies retrieving from the data source data to be published within pages. As already discussed in Section 3, the execution of adaptivity actions also requires the retrieval and evaluation of context meta-data. In WebML, a so-called `Get Data` unit (see Figure 6) has been introduced for retrieving values (both scalars and sets) from the data source, according to a selector condition. Its semantics is similar to the one of content publishing units, with the only difference that data retrieved from the data source are not published in a page, but just used as input to successive units or operations. It therefore represents a facility for accessing context meta-data stored in the application's data source whenever no visualization is required, for example in situations where certain data are just needed to evaluate condition expressions.

—*Updating the context model.* Once fresh context parameters have been retrieved, they must be used for updating the context model at data level. This action consists in modifying values previously stored in the data source. In WebML,

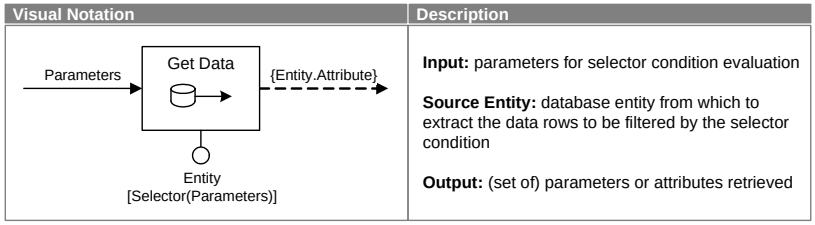

Fig. 6. Visual notation for the *Get Data* unit.

its specification uses operation units (see Section 4.1) providing support for the most common database management operations (`modify`, `insert`, `delete`).

—*Monitoring the context model.* This feature is achieved by introducing periodic checking mechanism for querying the context model and identifying any variation of the page context that requires the execution of adaptivity actions. Such check implies the periodic polling of `C`-pages and the evaluation of some conditions, as described in the following.

4.4.2 *Evaluating Conditions.* The execution of adaptivity actions may depend on the evaluation of some conditions. The most recurrent pattern consists of evaluating whether context has changed, hence triggering the adaptivity actions. The evaluation of conditions is specified through two control structures, represented by the `If` and `Switch` operation units that have been recently proposed for extending WebML for workflow modeling [Brambilla et al. 2003].

4.4.3 *Executing Adaptivity Actions.* Once the current context state is determined, and possible conditions are evaluated, adaptivity actions can be activated for customizing the page contents, the navigation, the current site view and the presentation style. Such 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 cloud. The use of parameterized selectors allows for both *filtering* data items with respect to the current context, and also conditionally including/excluding specific 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 to a different page. The specification of context-triggered navigations just requires connecting one of the links exiting the context cloud 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 page represent *automatic navigation actions*.

—*Adapting the Site View.* In some cases, a context-triggered switch toward a different site view is performed. 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 he/she shifted to a different activity. Therefore, we have introduced a `Change Site View` unit (see Figure 7), which takes

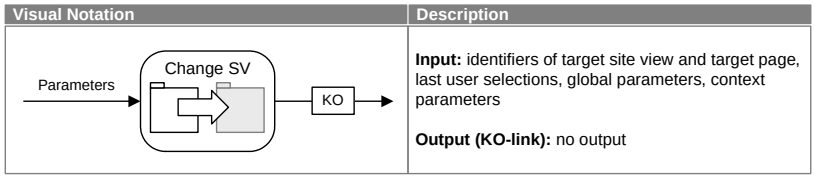

Fig. 7. Visual notation for the *Change SiteView* unit.

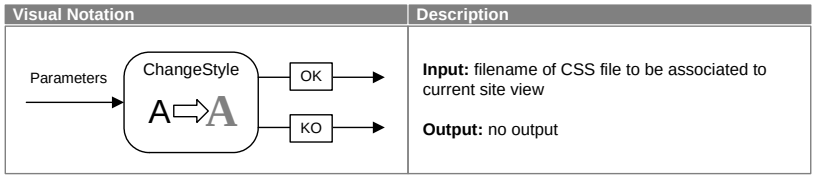

Fig. 8. Visual notation for the *Change Style* unit.

in input the identifiers of a target site view and a 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 OID and group OID), as well as some past user selections;
(3) Parameters characterizing the current context, retrieved through the latest performed data acquisition cycle and not yet stored persistently into the context model.

—*Adapting Presentation Style.* Sometimes context changes may require only fine-grained adaptations of presentation properties (i.e., due to varying luminosity conditions), not a complete restructuring of the overall hypertext. While adaptations of the page layout can be achieved by means of this latter approach or by conditionally including/excluding content units, we also defined a `Change Style` unit for dynamically selecting presentation style properties (see Figure 8). Style properties are collected in proper `.css` (Cascaded Style Sheet) files, and the unit allows changing at runtime the style sheet associated to the application.

5. A MODELING EXAMPLE

The modeling framework described so far has been conceived and applied within the Italian FIRB research project MAIS (*Multichannel Adaptive Information Systems* [MAIS Consortium 2005]). For validating the project results, an Integrated Tourist Information System (ITIS) has been considered as case study; it provides tourists with up-to-date, personalized and location-aware information. For presentation purposes, in this paper we present a simplified version of the overall system

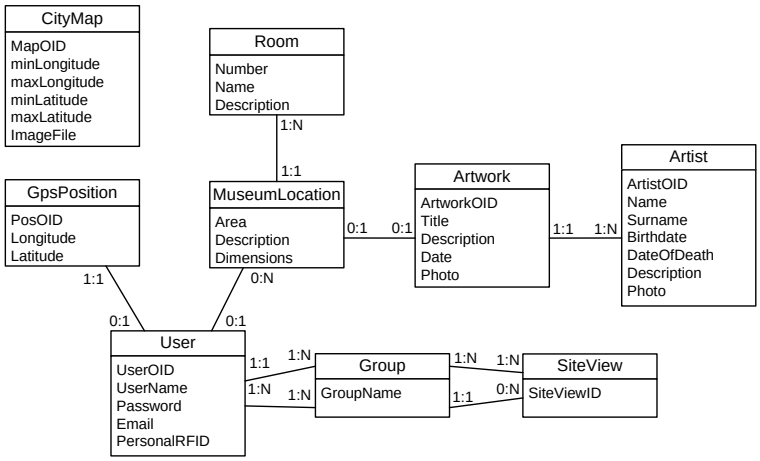

Fig. 9. Sample data schema for the Integrated Tourist Information System ITIS.

design[3]. By combining outdoor and indoor location mechanisms, ITIS allows for suitable coarse-grained and fine-grained information support. Outdoors, ITIS provides mainly city maps and sights descriptions as well as possible advertisements of nearby restaurants. Indoors, the available contents are defined autonomously by the administrators of the respective structures and vary from building to building. For example, the City Museum provides their visitors with location-aware descriptions of artworks and artists.

For the outdoor positioning of tourists, the system makes use of GPS (Global Positioning System) coordinates, whereas indoor sensing is achieved by means of active RFID tags. Each tourist is equipped with a PDA, a GPS module and a personal RFID tag, possibly integrated into the PDA. HTTP requests directed to ITIS are enriched with GPS coordinates by means of a small proxy server installed on the client PDA and managing the communication between PDA and GPS module. Indoor-sensed positions are directly fed into the application's context model by the sensing infrastructure.

Derived from this scenario, Figure 9 depicts the data source adopted by ITIS. Besides proper application data (entities `CityMap`, `Room`, `Artwork` and `Artist`), the entities `GpsPosition` and `MuseumLocation` build up the application's *context model*. While the GPS coordinates associated to users are updated by means of proper operations at hypertext level (see Figure 10), the association of locations inside the museum with users is maintained by the indoor sensing infrastructure, based on the `PersonalRFID` attribute of each user. More precisely, when a user enters one of the sensible areas in the museum, its `PersonalRFID` is revealed and the association area-user is updated.

The hypertext model of the ITIS Web application depicted in Figure 10 is built on top of the data source of Figure 9. The schema includes both `C`-pages and `C`-areas, and shows only context-triggered navigation alternatives. Each container

[3]For more details and a demo, see `http://dblambs.elet.polimi.it/Demos/`

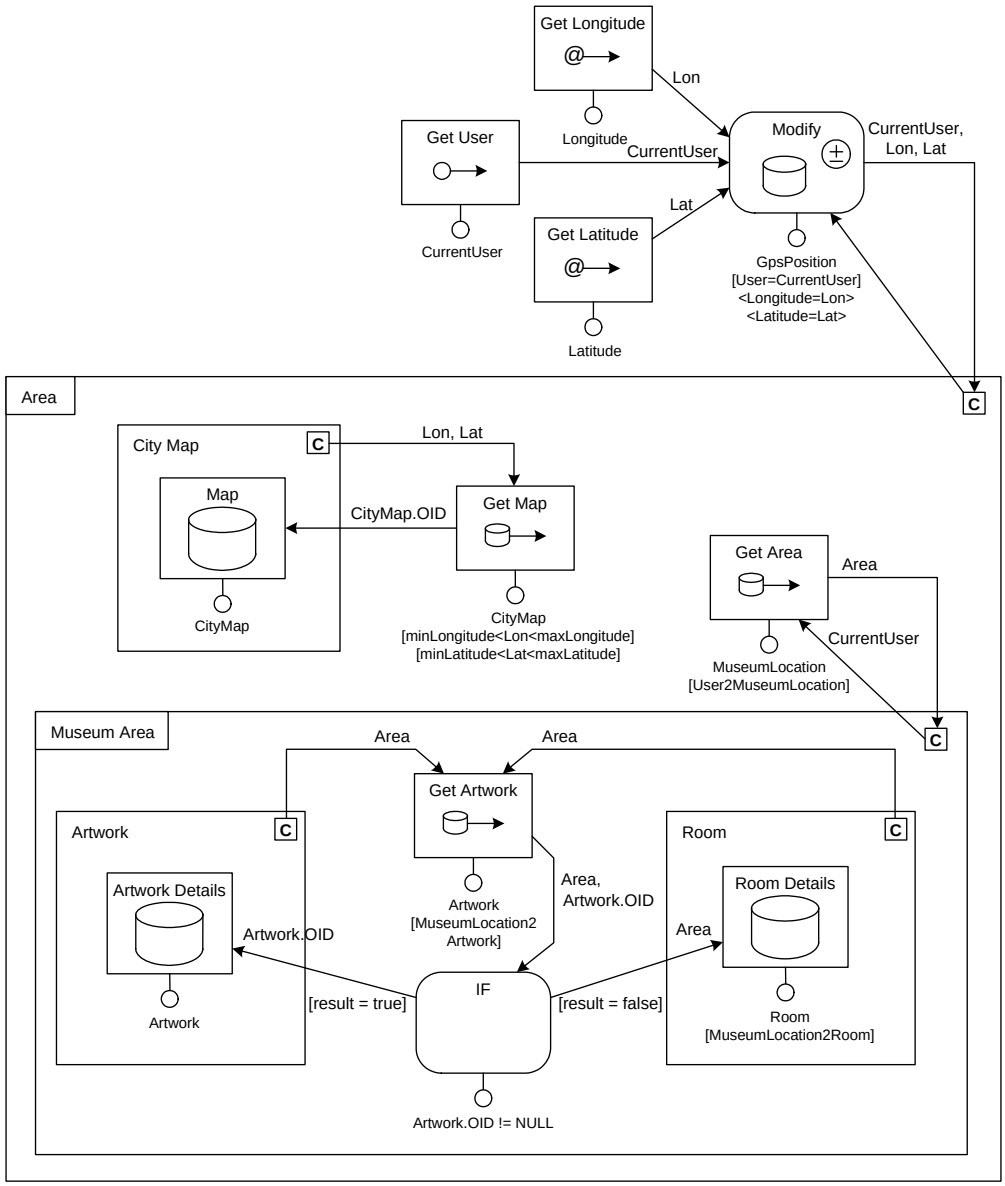

Fig. 10. Hypertext schema for the ITIS application. The schema only depicts context-related navigations (links), while user navigations are omitted for improving readability. Concurrency between user and context is discussed later on in this paper.

has its own chain of adaptivity actions, and some context parameters are passed hierarchically from outer context clouds to inner ones. For example, the parameters `Lon` and `Lat` (as well as the parameter `CurrentUser`) are generated by the outermost area and consumed by the cloud of the `City Map` page; also, the parameter `Area` is produced by the adaptivity actions associated to the `Museum Area` and is consumed for adapting the two pages `Artwork` and `Room`.

The outermost area is associated with actions for acquiring fresh context data and for updating the context model. More precisely, the current outdoor position of users is managed according to the following steps:

—The value of the session parameter `CurrentUser`, holding the current user ID, is retrieved through the `Get User` unit.

—The value of client-side parameters representing the current longitude and latitude are retrieved by means of the `Get Longitude` and `Get Latitude` units.

—The new GPS position is associated with the retrieved user by modifying the respective data entity (`Modify` unit).

These steps are executed each time the context clouds associated to one of the inner `C`-pages are evaluated. In order to guarantee the consistency of the context parameter passing mechanism, the steps are executed *before* any of the inner context clouds are evaluated. The `Adaptivity_Policy` property for both the area and its pages is set to `deferred`; therefore the context clouds, from the outermost to the page's one, are evaluated only when automatic refreshes following the user access are generated. In other words, when a user is viewing page `City Map`, before executing the adaptivity actions on page refresh, the outer cloud is executed and the parameters `CurrentUser`, `Lon`, and `Lat` are set. Only afterward, the page adaptivity actions that require the parameters `Lon` and `Lat` in input are performed, and a suitable city map is chosen for rendering the HTTP response.

On the other hand, the pages contained in the area `Museum Area` are built upon indoor position data, which are retrieved by means of the context cloud associated to the `Museum Area`. Also for this area, the `Adaptivity_Policy` is `deferred`. The only operation of the cloud consumes in input the parameter `CurrentUser` and produces in output the parameter `Area`, to be passed on to the inner pages.

The page `Artwork`, for example, uses the parameter `Area` for evaluating its adaptation rule that retrieves the ID of the artwork associated to the user's current position, checks whether the ID is not null (which would mean that no artworks exist in that position), and updates the content of the `Artwork` page if the retrieved ID contains a valid identifier. Otherwise, not being able to provide meaningful artwork details, the rule redirects the user to the page `Room` publishing details about the current exhibition room. The `Room` and `Artwork` pages share the same adaptivity actions, which trigger automatic navigations between the two pages in function of the availability of proper artwork data.

6. COMPUTING CONTEXT-AWARE PAGES

The introduction of context-awareness demands for a revision of the WebML page computation algorithm [Ceri et al. 2002a], which now needs to cope with the automatic execution of adaptivity actions associated to pages and containers.

The computation algorithm for conventional WebML pages is based on the computation of page-internal units. It starts by computing all the units that do not receive any link in input, and therefore do not need any parameter for their computation. Then, it proceeds with externally dependent units for which however there are sufficient input values in the parameters passed to the page. Hence, until all possible units inside the page have been computed, the algorithm iteratively selects the unit to be computed next, on the basis of the following conditions:

—All mandatory input parameters of the unit must have a value;
—All units that could supply a value to an input parameter of the unit must have already been computed.

When computing hypertext schemas supporting adaptivity, the page logic must be adapted for covering the automatic execution of adaptivity actions associated to both pages and their outer containers. Therefore, computation of adaptive pages must not only tackle the problem of how to compute the units contained in the page, but it also must guarantee the correct activation and execution of the associated context cloud. As already expressed by Figure 4, this implies recursively evaluating each context cloud before actual page computation, starting from the outermost one and up to the innermost one (i.e., the cloud associated to the page to be computed). Hence, the computation logic for ordinary pages keeps its validity, but it now handles also possible adaptation operations by recurring over context clouds defined for pages and their containers. This behavior can be summarized by the following `codeGen` function, where `buildContext` and `buildPage` perform the computation of the context cloud and the page, respectively:

```
FUNCTION codeGen(C:Container)
BEGIN
  IF (contextAware(C) THEN {
    IF (included(C,C') AND contextAware(C')) THEN
      codeGen(C');
    newPage = buildContext(C);
    IF (newPage != null) THEN
      codeGen(newPage);
  }
  IF (isPage(C)) THEN
    buildPage(C);
END
```

In particular, `buildContext` returns a value that, when the evaluation of the context cloud triggers an automatic navigation to a different page `newPage`, represents a pointer to the target page. This value is `null` when any automatic navigation is required. Also, the immediate or deferred activation of the context cloud computation within the `buildContext` function is subject to the adaptivity policy associated to the `C`-container under evaluation.

6.1 Specificity Rules

In some page configurations it may happen that a unit has multiple incoming links assigning values to the same parameter. Since only one value has to be considered,

the computation of the unit results to be ambiguous. Some *specificity rules* are therefore necessary for deciding which one of the incoming values to use.

For ordinary pages, the specificity of input parameters is assessed according to the following principles [Ceri et al. 2002a]:

(1) Values which derive from the current user's choice, expressed by the last navigation event, are the most specific;
(2) Values that depend on past user's choices or derive from global parameters accessed through `Get` units are the second most specific;
(3) Values heuristically deriving from the content of other units are the less specific.

In case of context-aware pages, the specificity rules are extended by means of a further condition to be evaluated *before* any other rule. Such a rule states that *values deriving from the computation of the page's context cloud (if evaluated) are the most specific*. The new specificity rule promotes context as a new actor that can cause navigation actions or page adaptations.

Coherent with this choice, the following classification shows the three possible situations that may occur when accessing pages:

—*Non-context-aware pages*: the ordinary specificity rules, not considering any adaptivity actions, apply, and units are computed as usual.
—*Access to C-labeled pages with deferred adaptivity*: adaptivity actions possibly defined for such pages are ignored at the first page access, in order to grant the *user* the highest priority. Possible adaptivity actions, specified for the page and its outer areas, are evaluated in response to automatic refreshes, periodically generated after the first user access to the page. This may result in overwriting previous user choices.
—*Access to C-labeled pages with immediate adaptivity*: the adaptivity actions are evaluated at each page request, also including the fist page access by the user.

Figure 11 illustrates the steps required at runtime for computing dynamic page templates, and highlights the additional computation required by context-aware pages. The use of a parameter `automatic`, appended to the HTTP page request, becomes necessary for identifying that a page has been requested by the automatic refresh mechanism and not by the user. As the figure shows, when the Web server receives a new page request, it decodes the incoming request parameters and, only for `C`-pages, it verifies whether adaptivity is needed. This check consists of (i) evaluating the value assigned to the `Adaptivity_Policy` page property and (ii) verifying the existence of the `automatic` parameter in the URL string. Accordingly, page computation proceeds as follows:

—If adaptivity is not required (i.e., for conventional pages or for the first user's access to `C`-pages with deferred policy), computation proceeds along the left hand side.
—If adaptivity is required, i.e., when `Adaptivity_Policy = ‘‘immediate’’`, or when `Adaptivity_Policy = ‘‘deferred’’` and parameter `automatic = ‘‘yes’’`, the computation proceeds along the right hand side. Two different adaptivity actions can be undertaken:

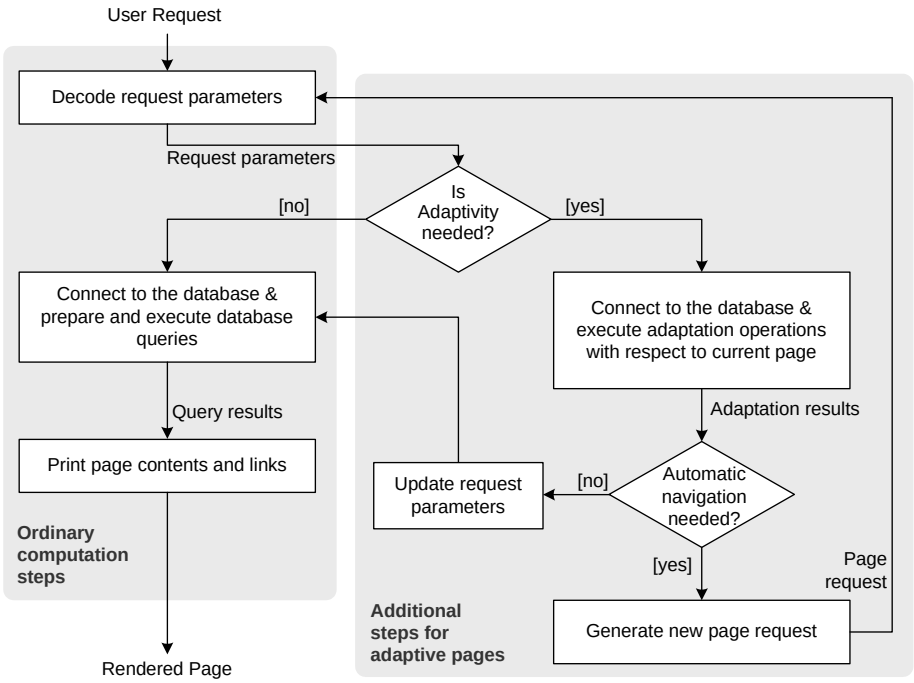

Fig. 11. Computation of context-aware page templates.

—*Page adaptation*: the database is accessed for reading the current state of the context, and some parameters are updated accordingly. Thus computation proceeds as with ordinary pages. The new value of context parameters will cause the adaptation of page content or style.
—*Navigation toward a different page*, within or outside the current site view: in this case, the computation process generates a new page request, and the page computation process starts anew, with the parameter `automatic = ‘‘no’’`.

It is worth noting that *infinite loops* with non-terminating evaluations of the context state could arise in the execution of chains of context clouds. This may occur when the target page of an automatic navigation starting from an “immediate” `C`-page adopts as well an immediate adaptivity policy and its context cloud is in conflict with the adaptivity actions specified for the source page. This in fact could redirect the user back to the source page, then again to the target page (due to the immediate policy), and so on.

The problem of non-termination is well-known in active databases (see, e.g., [Widom and Ceri 1996; Aiken et al. 1992; Baralis and Widom 1994]) and it is not surprising to find it applicable to adaptive Web computations; however, a sensible design of the Web application should rarely cause conflicts or non-termination. Design-time techniques can be used for checking either the acyclicity of page invocation graphs (a sufficient condition) or the lack of interference of cyclic page invocations (based upon semantics), along the directions marked in [Widom and Ceri 1996].

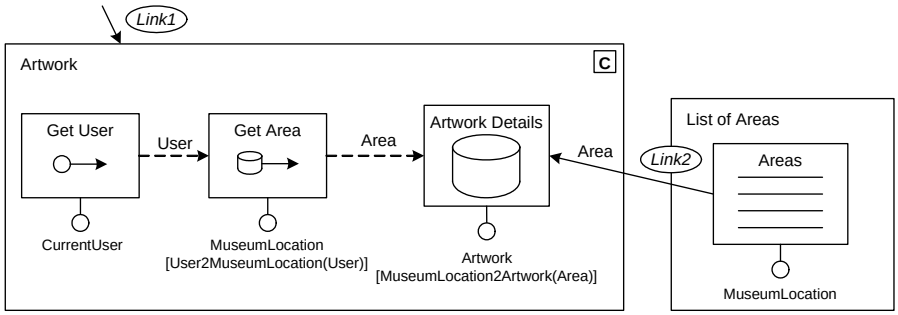

Fig. 12. Hypertext schema presenting either an adaptive or a non-adaptive behavior at runtime. `Link1` and `Link2` represent possible user navigations toward the context-aware page under investigation.

A design guideline for preventing infinite loops is to avoid cycles of automatic navigations involving source and target pages both with immediate adaptivity. When cycles need to be defined, a deferred policy for the involved pages is recommended. This ensures that the target page is rendered to the user before considering additional adaptivity actions. Therefore, the user can interrupt the (possible) cycle by disabling the context-aware modality or navigating to another page.

6.2 Context-Aware Page Computations

This section shows some examples of page computation with the objective of clarifying the interleaving and cross-effects between user navigation actions, automatically generated page requests, and adaptivity actions. The examples show that the page modeling logic needs to be well understood when pages are context-aware.

Consider the page in Figure 12, and assume it not to be "context-aware", in the sense that it does not reflect any change of context (therefore the figure has to be seen "without" the `C`-label); however, it reads context-specific meta-data, as represented by the `Get User` and `Get Area` units, respectively retrieving the current user, and its current museum area as shown by the `User2MuseumLocation` relationship. The page can be accessed along two links. `Link1` does not carry parameters to the page, while `Link2` carries as parameter an area, selected by the user by means of the area index shown in page `List of Areas`. The logic of the page, computed according to "standard" specificity rules, as described by items (1-3) of section 6.1, is to show the details of the artwork of the user-selected area if the page is accessed along `Link2`, and otherwise to show the artwork details for the user's current area. In particular, when `Link2` is traversed the specificity of the user-selected area prevails in the computation of the `Artwork Details` unit, and the page does not adapt its content to the current area where the user is located.

Consider then what happens if the page behaves as a context-aware page, as it is indeed represented by the `C`-mark in the figure. The following behavior occurs (independently from the chosen adaptivity policy):

—If the page is accessed along `Link1`, changes to the location of the user are reflected by a change of the artwork details being displayed; the page correctly adapts its content to the current area thanks to the `Get User` and `Get Area`

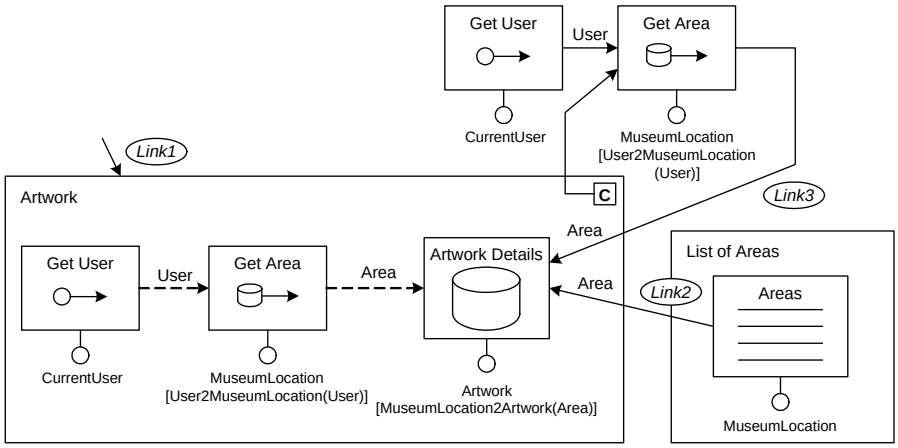

Fig. 13. Overwriting out-of-date link parameters. `Link1` and `Link2` represent user-navigated access to page `Artwork`. `Link3` transports fresh context data, retrieved within the context cloud, which overwrite past user choices.

units, able to read context data. At each refresh following the first user access, the page is also updated with respect to the current context.

—However, if the page is accessed along `Link2`, then the user-selected value (the most specific) prevails, and the page does not adapt its content. This value also prevails at each refresh.

The second behavior keeps the content of the page unchanged when the page is initially accessed by a link carrying a user selection; the designer could instead opt for a uniform behavior for both cases (and therefore, a "uniform adaptivity" regardless the navigated link). This uniform behavior can be achieved by redesigning the page so as to make the retrieval of the current area an explicit context-specific operation. Such evaluation should therefore be part of a context cloud, as indicated in Figure 13. In this way, adaptivity actions are given a higher priority with respect to past user selections. Assuming a deferred adaptivity policy, the page is then computed as follows:

—**Page access along Link1.** The page `Artwork` is accessed through a link which does not carry any parameters, and the page shows the details of the monuments in the area where the user is located. Being the first page access, this user-navigated link does not activate the adaptivity actions in the context cloud. Context data are however retrieved by means of the `Get Data` unit inside the page.

—**Page access along Link2.** The page `Artwork` is accessed through a link providing the area selected by the user. As in the previous case, this link does not trigger the context cloud. The page does not adapt its contents at all, because the parameter of `Link2` prevails over the `Area` parameter produces by the `Get Data` unit.

—**Page access through refresh.** The context cloud actions are executed. As a result, the current user area is passed in input to the unit `Artwork Details`

by using `Link3`. For the data unit to be computed, three values of the `Area` parameter are now available: the last user-selected area OID provided by `Link2`, the area OID retrieved by means of the get data unit internal to the page, and the area OID retrieved by the context cloud and provided by `Link3`. According to the specificity rules, values generated within the context cloud prevail over user-generated values; thus `Link3` "overwrites" `Link2`, regardless of the initial access to the page.

In case of an immediate adaptivity policy, values generated within the context cloud always prevail, because the context cloud is evaluated at each page access, regardless of the navigated link and the actor of the navigation (user versus automatic refresh).

The three examples discussed in this section show three possible types of context-awareness: *static* (for conventional pages, making use of Get units for retrieving the current values for context data), *dynamic with distinct refresh semantics based on the initial access* as represented in Figure 12, and *dynamic with uniform refresh semantics* as represented in Figure 13. Each of them is applicable in given cases, and the designer should choose the most appropriate one.

7. IMPLEMENTATION EXPERIENCES

In the context of the MAIS project, model extension, code generation, and runtime experiments were conducted, allowing us to successfully cover all aspects of the outlined modeling approach. Prototype development proceeded by two complementary steps; the first produced an external proof-of-concepts extension of the WebML runtime environment, the latter finally yielded to an (internal) extension of the WebML CASE tool *WebRatio* [WebModels s.r.l. 2005].

The external solution fully reflects the page computation logic outlined in Figure 11 and builds on a pre-processing mechanism for page requests. Within the MVC (*Model View Controller* [Davis 2001]) architecture of the WebML runtime environment [Ceri et al. 2003], a modified application *controller* accepts arbitrary HTTP requests and checks whether they refer to context-aware pages (as indicated by the presence of the request parameter `automatic` – see Section 6.1) or to conventional ones. In case of requests for conventional pages, the controller simply proceeds with page content computation; in case of context-aware pages, it first performs the associated adaptivity actions and then proceeds with the computation of page contents. Controller and adaptivity actions are implemented by hand, and allow managing (i) access to context data, (ii) adaptivity of page contents by overwriting request parameters, and (iii) automatic navigation actions by generating substitutive page requests. Controller logic and page-specific adaptation logic are intermixed and hardwired within a dedicated Java servlet in charge of pre-processing HTTP requests directed to the Web application.

This first proof-of-concepts prototype allowed us to assess the feasibility of the proposed conceptual modeling solutions, to set up an experimentation environment for simulating and testing their viability and, finally, to improve some minor aspects. The external adaptation solution takes full advantage of the existing WebML runtime environment by pre-processing its input, but lacks support for visual design and automatic code-generation starting from adaptive WebML schemas.

The previous two aspects are addressed by the second prototype that consists in the extension of the WebRatio tool to fully reflect the proposed (visual) design method. The implementation exploits WebRatio's native extension mechanisms that allow adding new features by means of so-called *custom units*, a mechanism that already has demonstrated its power when extending the CASE tool to support communications with Web services [Manolescu et al. 2005] and workflow-driven hypertexts [Brambilla et al. 2003].

In particular, the extension occurred along two complementary dimensions: the first dimension referred to the new adaptivity actions described in Section 4.4 (to be applied in the context cloud), while the second dimension concentrated on the introduction of context-aware pages as described in Section 4.3 (context-aware areas or site views are not supported yet). Novel operations defined as custom units are: the `Get ClientPar` unit[4], the `Get Data` unit, the `Change Site View` unit, and the `Change Style` unit. The introduction of context-aware pages required an extension of the page logic, yielding a further new content unit (called `Context` unit), to be used in place of the `C`-label associated to context-aware pages and triggering the context cloud logic. This unit also contains the parameter passing logic and manages the polling mechanism, granting the context cloud control when required, according to the chosen adaptivity policy.

With respect to the first experiments conducted, this second prototype implementation finally allows us:

—To support the design of arbitrary context-aware Web applications and the automatic generation of the respective program code;

—To fully reflect the adaptive design method proposed in this paper;

—To capitalize on the WebML CASE tool and runtime environment.

The screenshot in Figure 14 refers to the extended visual WebRatio environment, and shows a WebML model fragment of the location-aware ITIS Web application described in Section 5. Context-aware pages contain the aforementioned `Context` unit, which takes in input the current user's longitude and latitude (accessed by means of two `Get ClientPar` units) and forwards them to the single pages' context clouds in case adaptivity actions must be triggered. We omit a detailed description of the single modeling constructs, and rather focus the reader's attention on the following two peculiarities: (i) City map data are retrieved by means of an external Web service[5]; the operations outside the `Assistant` area manage the respective communications. (ii) Since proper context-aware containers, i.e. areas, are not supported yet by the current version of WebRatio, typical area-level adaptivity actions (i.e., accessing longitude and latitude) are repeatedly specified as page-level operations. Support for context-aware containers is planned for future versions of the WebML CASE tool.

[4]In the current implementation, the communication of client-side sensed data occurs by appending parameters to the URL of the requested page. Other communication mechanisms, also based on the SOAP communication protocol, are planned for future releases.

[5]Microsoft MapPoint Web Service, `http://www.microsoft.com/mappoint/products/webservice/default.mspx`

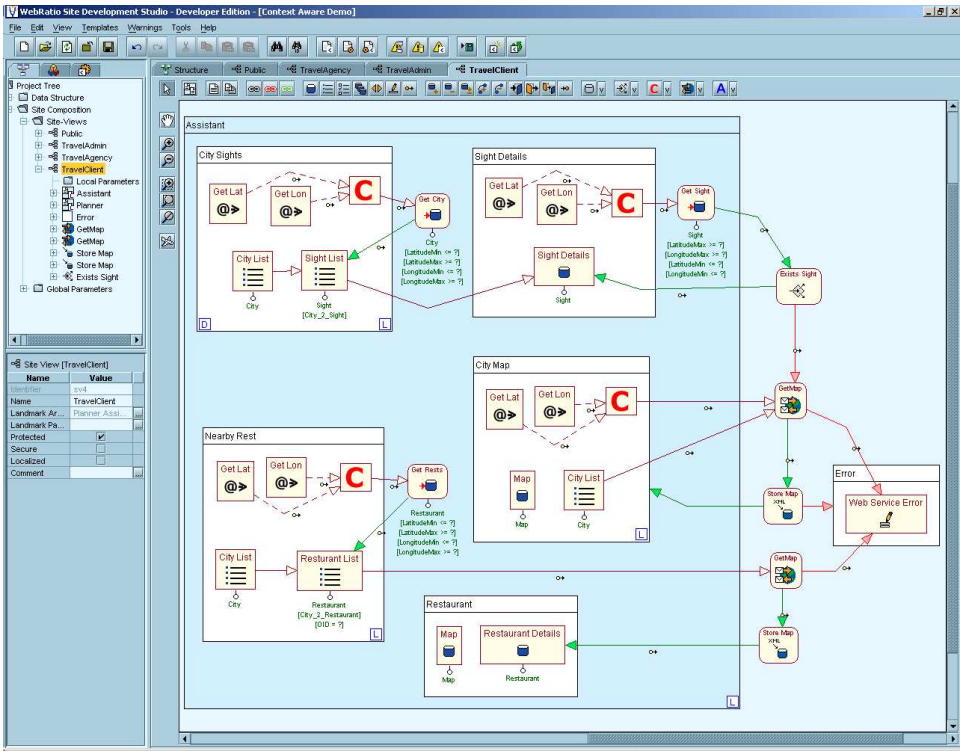

Fig. 14. Screenshot of the visual environment of the WebML CASE tool, showing a fragment of the location-aware ITIS Web application design.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have considered a relevant aspect of modern Web applications, i.e. adaptability to context, and we have shown how such aspect requires increasing the expressive power of Web application models so as to incorporate changes in the page generation logic that depend on the context. The proposed approach, based on WebML, enables a fully automatic generation of adaptive applications. However, the proposed design primitives are general in nature, and context modeling, adaptation changes, parameter passing, and the use of refresh, can be manually encoded by Web programmers.

The solutions described in this paper have been extensively tested in the context of the MAIS project, by altering the run time component of WebRatio; this proved 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 (first quarter 2006).

As a continuation of this research, we are now defining a formal model for analyzing possible conflicts and inconsistencies that may be caused by the activation of adaptivity actions. We are also studying how device-specific Web services could be published (by using the new "service view" concept of WebML [Brambilla et al. 2005]), so as to enable the model-driven specification of meta-data management for

context data. Additionally, we are studying adaptability modes, which are either time-dependent (i.e., whose refresh delay can be modeled according to given policies) or real-time (i.e., whose triggering is immediate thanks to client-side aware extension).

Further efforts will investigate the potential of post-processing mechanisms for fine-grained adaptation of presentation properties, i.e., adaptive link hiding, and study techniques for non-invasive refresh management. Especially, the adoption of Rich Internet Application (RIA) technologies [Macromedia Inc. 2003; Laszlo Systems 2005] in this regard will enable a background communication between the client devices and the application server, useful for querying the context state and refreshing pages only when required. This will lead to an "active" context-awareness [Ceri et al. 2005] stressing the importance of context monitoring as a mechanism operating autonomously and transparently in the background, thus providing active support. The expected result is an improvement of the overall application efficiency and usability. In order to verify the usability of the proposed adaptivity paradigm, we are planning some experiments involving real users. The aim is to investigate the right balance between user-controlled interactions and context-triggered adaptations.

ACKNOWLEDGMENTS

This work has been supported by the Italian FIRB Project MAIS (Multi-channel Adaptive Information Systems).

REFERENCES

Aiken, A., Widom, J., and Hellerstein, J. M. 1992. Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism. In *Proc. of the SIGMOD Conference, 1992*. ACM, 59–68.

Baralis, E. and Widom, J. 1994. An Algebraic Approach to Rule Analysis in Expert Database Systems. In *Proc. of the VLDB Conference, 1994*. Morgan Kaufmann, 475–486.

Barna, P., Houben, G.-J., and Frasincar, F. 2004. Specification of Adaptive Behavior Using a General-Purpose Design Methodology for Dynamic Web Applications. In *AH'04 - Proc. of Adaptive Hypermedia*. 283–286.

Belotti, R., Decurtins, C., Grossniklaus, M., Norrie, M. C., and Palinginis, A. 2004. Interplay of Content and Context. In *ICWE*. 187–200.

Brambilla, M., Ceri, S., Comai, S., Fraternali, P., and Manolescu, I. 2003. Specification and Design of Workflow-Driven Hypertexts. *Journal of Web Engineering 1,* 2 (April), 1–100.

Brambilla, M., Ceri, S., Fraternali, P., Acerbis, R., and Bongio, A. 2005. Model-driven Design of Service-enabled Web Applications. In *Proc. of the 2005 SIGMOD Conference, 2005*. ACM, 851–856.

Brusilovsky, P. 1996. Methods and Techniques of Adaptive Hypermedia.. *User Model and User-Adapted Interaction 6,* 2-3, 87–129.

Ceri, S., Daniel, F., Facca, F. M., and Matera, M. 2005. Model-driven Engineering of Active Context-Awareness. Technical Report 11.9.2005, Politecnico di Milano. Available at `http://dblambs.elet.polimi.it/DBLambs/Publication.php?Publication_OID=38`.

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

Ceri, S., Fraternali, P., Bongio, A., Butti, S., Acerbis, R., Tagliasacchi, M., Toffetti, G., Conserva, C., Elli, R., Ciapessoni, F., and Greppi, C. 2003. Architectural Issues and Solutions in the Development of Data-Intensive Web Applications. In *Proceedings of CIDR 2003, January 2003, Asilomar, CA, USA*.

Ceri, S., Fraternali, P., and Matera, M. 2002. Conceptual Modeling of Data-Intensive Web Applications. *IEEE Internet Computing 6,* 4 (July-August), 20–30.

Ceri, S., Fraternali, P., and Paraboschi, S. 1999. Data-Driven One-To-One Web Site Generation for Data-Intensive Applications. In *Proc. of VLDB'99*. Morgan Kaufmann.

Davis, M. 2001. Struts, an Open-source MVC Implementation. `http://www-106.ibm.com/developerworks/library/j-struts/?n-j-2151`.

De Bra, P., Aerts, A., Berden, B., de Lange, B., Rousseau, B., Santic, T., Smits, D., and Stash, N. 2003. AHA! The Adaptive Hypermedia Architecture. In *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*. 81–84.

De Bra, P., Houben, G.-J., and Wu, H. 1999. AHAM: a Dexter-based Reference Model for Adaptive Hypermedia. In *HYPERTEXT '99: Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots*. 147–156.

Dey, A. K. and Abowd, G. D. 2000. Towards a Better Understanding of Context and Context-Awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000), The Hague, The Netherlands*.

Fiala, Z., Hinz, M., Houben, G.-J., and Frasincar, F. 2004. Design and Implementation of Component-based Adaptive Web Presentations. In *ACM SAC*. 1698–1704.

Fraternali, P. 1999. Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. *ACM Computing Surveys 31,* 3 (September), 227–263.

Garzotto, F., Paolini, P., and Schwabe, D. 1993. HDM - a Model-based Approach to Hypertext Application Design. *ACM Trans. Inf. Syst. 11,* 1, 1–26.

Grossniklaus, M. and Norrie, M. C. 2002. Information Concepts for Content Management. In *WISE Workshops*. 150–159.

Hansen, F. A., Bouvin, N. O., Christensen, B. G., Grønbæk, K., Pedersen, T. B., and Gagach, J. 2004. Integrating the Web and the World: Contextual Trails on the Move. In *Proc. of ACM-Hypertext'04*. 98–107.

Henricksen, K. and Indulska, J. 2004. Modelling and Using Imperfect Context Information. In *PerCom Workshops*. 33–37.

Henricksen, K., Indulska, J., and Rakotonirainy, A. 2002. Modeling Context Information in Pervasive Computing Systems. In *Pervasive*. 167–180.

Hjelm, J., Martin, B., and King, P. 1998. WAP Forum - W3C Cooperation White Paper. `http://www.w3.org/TR/NOTE-WAP`.

Laszlo Systems Inc. 2005. OpenLaszlo - an XML Framework for Rich Internet Applications. Laszlo Systems Technology White Paper.

Isakowitz, T., Stohr, E. A., and Balasubramanian, P. 1995. RMM: a Methodology for Structured Hypermedia Design. *Commun. ACM 38,* 8, 34–44.

Kappel, G., Proll, B., Retschitzegger, W., and Schwinger, W. 2003. Customization for Ubiquitous Web Applications - A Comparison of Approaches. *International Journal of Web Engineering and Technology*.

Kobsa, A., Koenemann, J., and Pohl, W. 2001. Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships. *The Knowledge Engineering Review 16,* 2.

Koch, N., Kraus, A., and Hennicker, R. 2001. The Authoring Process of the UML-based Web Engineering Approach. In *First International Workshop on Web-oriented Software Technology (IWWOST01)*, D. Schwabe, Ed.

Lei, H., Sow, D. M., II, J. S. D., Banavar, G., and Ebling, M. 2002. The Design and Applications of a Context Service. *Mobile Computing and Communications Review 6,* 4, 45–55.

Long, S., Kooper, R., Abowd, G. D., and Atkeson, C. G. 1996. Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study. In *MOBICOM*. 97–107.

Macromedia Inc. 2003. Developing Rich Internet Applications with Macromedia MX 2004. Macromedia White Paper.

MAIS Consortium. 2005. MAIS Project Home Page. `http://www.mais-project.it/`.

Manolescu, I., Brambilla, M., Ceri, S., Comai, S., and Fraternali, P. 2005. Model-Driven Design and Deployment of Service-Enabled Web Applications. *ACM TOIT 5,* 3 (August), In print.

Ni, L. M., Liu, Y., Lau, Y. C., and Patil, A. P. 2004. LANDMARC: Indoor Location Sensing Using Active RFID. *Wireless Networks 10,* 6, 701–710.

Salber, D., Dey, A. K., and Abowd, G. D. 1999. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proc. of CHI'99*. 434–441.

SALTforum.org. 2005. Speech Application Language Tags (SALT). `http://www.saltforum.org/`.

Schmidt, A., Aidoo, K. A., Takaluoma, A., Tuomela, U., Laerhoven, K. V., and de Velde, W. V. 1999. Advanced Interaction in Context. In *HUC*. 89–101.

Schwabe, D., Guimaraes, R., and Rossi, G. 2002. Cohesive Design of Personalized Web Applications. *IEEE Internet Computing 6,* 2 (March-April), 34–43.

Schwabe, D., Rossi, G., and Barbosa, S. D. J. 1996. Systematic Hypermedia Application Design with OOHDM. In *HYPERTEXT '96: Proceedings of the seventh ACM conference on Hypertext*. ACM Press, New York, NY, USA, 116–128.

Vdovjak, R., Frasincar, F., Houben, G.-J., and Barna, P. 2003. Engineering Semantic Web Information Systems in Hera. *Journal of Web Engineering 2,* 1-2, 3–26.

W3C. 2004. Voice Extensible Markup Language (VoiceXML) Version 2.0. `http://www.w3.org/TR/2004/REC-voicexml20-20040316/`. W3C Recommendation.

Want, R., Hopper, A., Falcao, V., and Gibbons, J. 1992. The Active Badge Location System. *ACM Trans. Inf. Syst. 10,* 1, 91–102.

WebModels s.r.l. 2005. WebRatio Site Development Studio. `http://www.webratio.com`.

Widom, J. and Ceri, S. 1996. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann.

Received Month Year; revised Month Year; accepted Month Year