

# Extending WebML for modeling multi-channel context-aware Web applications

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

## Abstract

*This paper focuses on some issues related to the conceptual modeling of multi-channel, context aware Web applications, and in particular it proposes some solutions conceived within the WebML method. WebML is a conceptual model for data-intensive Web applications, which already offers some constructs for one-to-one personalization and multi-channel delivery. In this paper we introduce some new extensions that will allow representing a context model at data level, and exploit it at hypertext level for offering customized services and contents, accessible through multiple channels.*

## 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 will change 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 environment. Due to such premises, the traditional design methods will not be anymore exhaustive, since new issues and requirements need to be addressed for supporting multi-channel, context-aware access to services. This paper focuses on the emerging new requirements, and proposes some solutions conceived within the *WebML* method.

WebML (Web Modeling Language) is an already established conceptual model for data-intensive Web applications, which is accompanied by a development method and a CASE tool [3, 4, 9]. WebML, and in general conceptual modeling, have already proven their effectiveness for the design of personalized Web applications [5, 8]. In this paper we illustrate how some extensions can also support the design of applications able to adapt themselves to the variability of the adopted communication channel and of the usage context.

The paper is organized as follows. Section 2-4 provides a short overview of WebML, recalling its basic concepts and notations and illustrating how the model already supports the specification of one-to-one personalization and multi-channel delivery. Section 5 introduces some new emerging requirements for the design of context-aware applications, and clarifies some

assumptions at the basis of our approach. Section 6 and 7 then illustrate the proposed extensions along two design dimensions: data and hypertext modeling. Section 8 presents the extended WebML at work for the specification of a museum application supporting users guided tours through exposed artworks. Section 9 finally draws some conclusions.

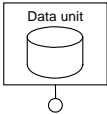
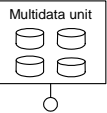
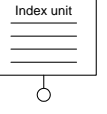
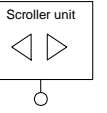
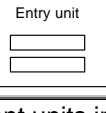
## 2. WebML: an overview

WebML is a visual language for specifying the content structure of a Web application and the organization and presentation of contents in one or more hypertexts.

The design process starts with the definition of a data schema, expressing the organization of the application content. The *WebML Data Model* adopts the Entity-Relationship (ER) primitives for representing the organization of the application data. Its fundamental elements are therefore *entities*, defined as containers of data elements, and *relationships*, defined as semantic connections between entities. Entities have named properties, called attributes, with an associated type. Entities can be organized in generalization hierarchies, and relationships can be restricted by means of cardinality constraints.

The *WebML Hypertext Model* allows then describing how data, specified in the data schema, are published into the application hypertexts. The overall structure of hypertexts is defined in terms of site views, areas, pages and content units. A *site view* is a particular hypertext, designed to address a specific set of requirements. It consists of *areas*, which are the main sections of the hypertext, and comprises 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 and published within pages. Content units and pages are interconnected by *links* to constitute site views.

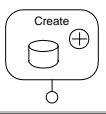
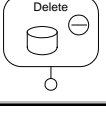
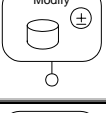
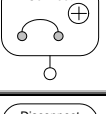
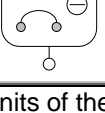
Several site views can be defined on top of the same data schema, for serving the needs of different user communities, or for arranging the composition of pages to meet the requirements of different access devices like PDAs, smart phones, and similar appliances.

<b>Data unit</b>		It displays a set of attributes for an entity instance.
<b>Multidata unit</b>		It displays all the instances for a given entity.
<b>Index unit</b>		It displays list of properties, also called <i>descriptive keys</i> , of a given set of entity instances.
<b>Scroller unit</b>		It represents a scrolling mechanism, based on a block factor, for the elements of a set.
<b>Entry unit</b>		It displays a form for collecting input values into fields.

**Table 1.** Content units in the WebML composition model.

The WebML Hypertext Model includes:

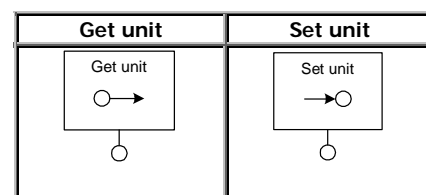
- The *composition model*, concerning the definition of pages and their internal organization in terms of elementary pieces of publishable content, called *content units*. Content units offer alternative ways of arranging content dynamically extracted from entities and relationships of the data schema. WebML units (see Table 1 for their visual notation) denote one or more instances of the entities of the data schema, typically selected by means of queries over entities, attributes, or relationships, and also forms for collecting input values into fields. Unit specification (excepting for the *entry unit*) includes the indication 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 predicate, used for determining the *actual objects* of the source entity that contribute to the unit's content.
- The *navigation model*, based on the definition of links that connect units and pages, thus forming the hypertext. Links can connect units in a variety of legal configurations, yielding to composite navigation mechanisms. Links between units are used to carry some information (called *context*) from the source unit to the destination unit.

<b>Create unit</b>		It specifies the creation of an entity instance.
<b>Delete unit</b>		It specifies the deletion of entity instances.
<b>Modify unit</b>		It specifies the updating of entity instances.
<b>Connect unit</b>		It specifies the creation of a relationship instance.
<b>Disconnect unit</b>		It specifies the deletion of a relationship instance.

**Table 2.** Basic units of the WebML operation model.

- The *operation model*, consisting of a set of units for specifying content management operations. The basic primitives for expressing update operations can be specified as reported in Table 3: they allow creating, deleting or modifying an instance of an entity (respectively represented through the *create*, *delete* and *modify* units), or adding or dropping a relationship between two instances (respectively represented through the *connect* and *disconnect* units).

WebML also provides units for the definition of global parameters. Parameters can be set through the *set* unit, and consumed within a page through a *get* unit. The visual representation of such two units is reported in Table 3.



**Table 3.** Visual notation of get and set units.

Besides having a visual representation, WebML primitives are also provided with an XML-based textual representation, used to specify additional detailed properties, not conveniently expressible in the graphic notation. Web application specifications based on WebML can be therefore represented as visual diagrams, as well as XML documents.

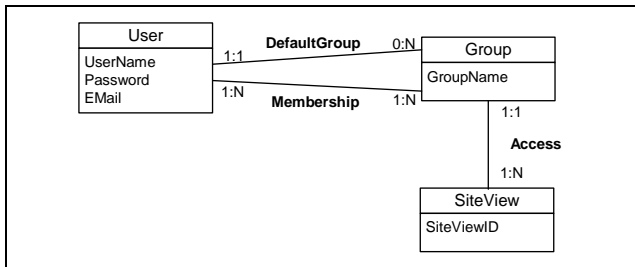


Figure 1. WebML user model.

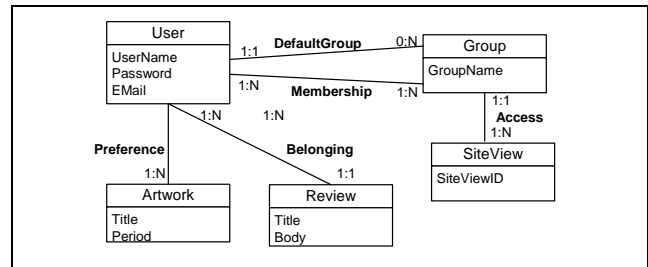


Figure 2. Data schema for a personalized application.

### 3. WebML and one-to-one personalization

Several application requirements impose that a given user accesses a sub-set of data and functionality, according to his/her own user profile. The WebML method supports personalization of contents and services, assuming the key principle that *users and their roles must be modeled as data*. At this aim, the WebML data schema of a Web application typically includes entities representing users and the groups in which they are clustered.

Additionally, the delivery of different viewpoints and functions to selected user groups requires designing alternative WebML site views and “attaching” each site view to the user group for which it has been designed. Such an association can be obtained by modeling also site views as data, for example introducing into the data schema a *SiteView* entity, and connecting this entity to the *Group* entity by means of a relationship. In this way, during application execution, it is possible to forward the user to the home page of the appropriate site view, based on the group membership.

The essential aspect of the WebML approach to one-to-one personalization is therefore the inclusion of users, groups and site views as “first-class citizens” in the application data schema. The data schema of every application designed with WebML thus includes a default sub-schema, which is showed in Figure 1:

1. The entity *User* includes the basic properties of each user. The entity *Group* includes collective properties.
2. A many-to-many relationship (called *Membership*) connects *User* to *Group*, denoting that a user may belong to multiple groups, and that a group clusters multiple users.
3. A many-to-one relationship (called *DefaultGroup*) connects *User* to *Group*, denoting that a user may have one group as the default one among the groups he belongs to. This additional information is useful for assigning the user to the default group after he logs into the application, and for forwarding him to the site view of the default group.

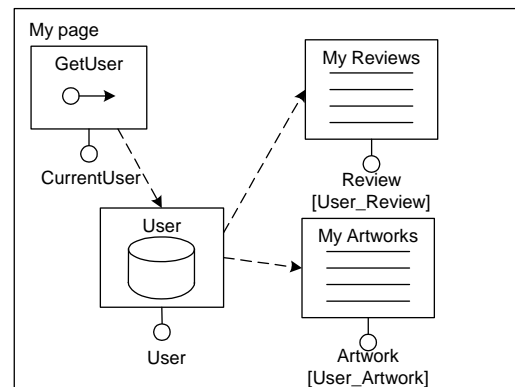


Figure 3. Hypertext schema of a Web page delivering personalized content.

The described user model applies also to those Web applications in which users remain anonymous, but are nonetheless individually tracked, for example in virtue of their IP addresses or session identifier. In this case, anonymous users can be considered as instances, although temporary, of the *User* entity. This permits to provide them with personal data for the duration of the session, for example with an individual trolley, even though their credentials are unknown.

The definition of relationships between the entity *User* (or *Group*) in the user model, and some other information objects in the application content makes it possible modeling *preferences* over any information object, and *ownership* over objects created and managed by individual users. For example, in Figure 2 the application entities *Artwork* and *Review* are associated to individual users through relationships expressing the user preferences over artworks and user participations in writing reviews. On this schema it is possible to specify the personalized hypertext of Figure 3. By means of the global parameter *CurrentUser*, representing the identifier of the current user, the page publishes exactly some data about the current users plus the list of artworks preferred by the users and the list of reviews s/he has written about artworks.

## 4. WebML and multi-channel delivery

Site views may also serve the purpose of expressing alternative forms of content presentation on different devices. Each site view may cluster information and services at the granularity most suitable to a particular class of devices.

For example, one application can feature two different hypertexts defined over the same content: one for PC browsers, the other for WAP devices. Given the difficulty of browsing with a small screen, the WML version must be defined so as to provide very concise contents and minimized interactions.

Hypertext modeling can be therefore applied to multi-channel applications, by devising the site view structures most suited to the specific delivery medium, that take into account the specific requirements of each device with respect to the amount of information that can be placed within pages and the complexity of navigation.

It is worth noting that WebML requires the definition of distinct sit views only when devices show very different rendering capabilities, and therefore require completely different hypertext structures. For devices with similar features, at design time it is possible to model one only hypertext, and then translate it into the multiple mark-up languages required by devices. Once the site view schemas for the different devices are established, producing the actual pages in the proper markup language is a matter of implementation, generally based on the definition of XSL style sheets for the production of markup code.

## 5. WebML and context-awareness

Context-awareness is often seen as a recently emerged research field within information technology. From another perspective, it can be however interpreted as just an extension of personalization, addressing not only the user's identity and preferences but also the interaction environment that hosts applications.

Context can be described in terms of properties 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 needed adaptation [6, 7]. Besides users data, adaptive solutions for context-aware applications thus exploit context data for providing users with more useful and usable services, in a scenario where, especially due to the mobility and multiplicity of devices, the context can vary.

Context-awareness therefore requires automatism on the application part that are triggered by changes of anyone of the parameters that make up context.

## 5.1 Functional requirements

In this paper we focus on WebML and its extensions for supporting the specification of context-aware applications. WebML supports data and hypertext modeling. Thus we will introduce some new concepts along these two dimensions, which allow representing the context status at data level, and also specifying some reactive behaviors of the application with respect to the context state.

Since the WebML modeling approach is data-driven, the use of context within application primarily requires its representation at data level. However, also mechanisms for context data acquisition and access are needed [7]. Therefore, a set of the envisioned extensions allows fulfilling the following goals:

- *Context model representation* through a dedicated data sub-schema;
- *Acquisition of context data* made available by the usage environment;
- *Continuous update of the context model* within the application data source, for reflecting the newly reached context state.

Some other goals are concerned with adaptivity actions over the application hypertext:

- *Adaptivity of currently visited page*, based on a new reached context.
- *Adaptivity of navigation*, through automatic navigation<sup>1</sup> towards pages, within a same site view, which are considered more appropriate for the reached context.
- *Adaptivity of the whole hypertext structure*, by means of site views switching, for facing changes of the user's device, role or activity.

## 5.2 Architectural requirements

In order to reach the previous goals, it is necessary to monitor some context data, and capture them from the environment in which the application is executed. Figure 4 shows the context data flow within a possible architecture tailored to support context-aware hypertext solutions. It is composed of two layers related to the application data source and hypertext. 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

---

<sup>1</sup> By automatic navigation we mean jumps to pages driven by the application, not by the user actions.

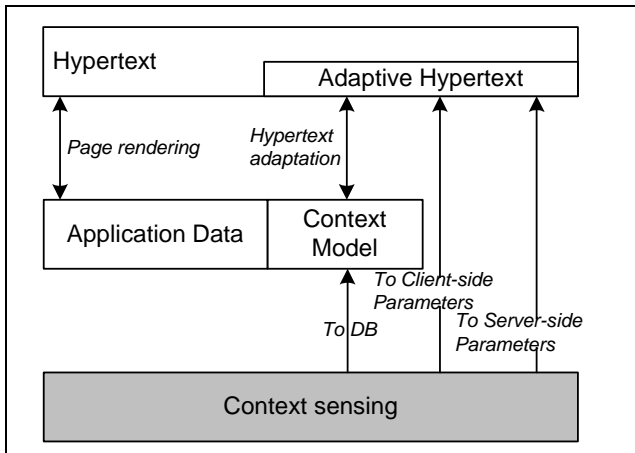


Figure 4. Context Data Flow.

call *context model*. The hypertext computation primarily exploits the application data for page and unit computation. We then assume that only a subset of hypertext elements, included in the so-called *adaptive hypertext*, is augmented with adaptive behaviors. The computation of such elements will exploit also context data. Moreover, the adaptive hypertext includes the invocation of operations that, on the basis of newly sensed context data, update the context model.

As expressed by the gray box, the proposed architecture also features context-sensing mechanisms for capturing context data from the application execution environment. Since context sensing is a technological issue, which does not affect the specification of the final hypertext, we will not focus on it. We only assume that some solutions, for example a sensor infrastructure, allow sensing such data. We then identify three possible mechanisms for context data acquisition and representation:

- As parameters generated by client-side mechanisms and communicated back to the application through the URL of requested pages;
- As session parameters, generated at server-side.
- As data retrieved from the context model in the application data source.

While the first two mechanisms act at the hypertext level, providing session variables used for computing the adaptive hypertext, the last acquisition mechanism acts at data level. As it will be better described in the following sections, some dedicated operations, invoked within the adaptive hypertext, will serve the purpose of keeping the context model updated on the basis of fresh captured values. However, we can also assume that asynchronous services are used for this purpose.

The reminder of the paper describes the WebML extensions that are required to support context awareness.

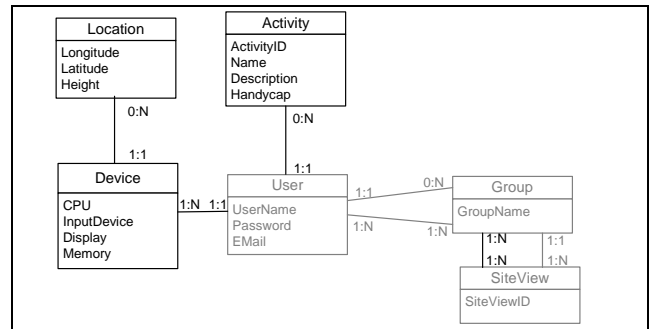


Figure 5. Context sub-schema.

Similarly to other proposed extensions of WebML (e.g., to process management [1] or to Web services [2]), the extensions are performed by (i) defining a suitable data schema for modeling context (Section 6) and (ii) defining model extensions for managing adaptivity (Section 7).

## 6. Modeling context through data sub-schemas

The modeling approach proposed by WebML is data-driven. We therefore propose to enrich the application data schema with a *context model*, that is the representation of metadata needed for supporting adaptivity. By explicitly representing these concepts as entities, many useful customization policies can be expressed declaratively in the data and in the hypertext schema, instead of being buried in the source code of the application.

The context model can vary depending on the application domain, and also on the adaptivity goals to be fulfilled. For this reason, we will not provide a precise, rigid characterization of context. Rather, we will introduce some guidelines about how to extend the data schema through context metadata.

Figure 5 shows a possible sub-schema for representing context information, which extends the user model sub-schema illustrated in Section 3. It is possible to note two main extensions, consisting of elements represented with black borders: the *context entities*, for representing context data, and a new relationship between the entities *Group* and *SiteView*.

The context entities are associated to each user. In Figure 5, they for example represent the user device and its location, and the user activity. Some applications may also require a different set of entities for representing the context. However, our approach just prescribes to have such entities associated (directly or indirectly) to the User entity, which is the starting point in the schema for navigating within the context model and extracting context information.

The new many-to-many relationship between Group and Site View represents all the site views associated to a given group, among which users can switch during the application execution, according to the adaptivity rules defined in correspondence of context changes. With respect to the WebML user model (see Section 3), this new relationship expresses that users belonging to a given group can however act in different contexts.

Site view switching can therefore occur when the user change devices, thus a new hypertext for accessing contents is required - this allows capturing multi-channel requirements. Also, switching is needed when contents and services to be provided in a new context are totally different, thus a new application view is required. For example, a museum application could provide users with completely different hypertexts when they are visiting artworks inside the museum building using a PDA for accessing artwork description, and when they are accessing the application with the same device outside the building, for getting information about the current expositions and the opening hours.

The one-to-many relationship remains unchanged with respect to the WebML user model. In the context model it however represents the association of a group with a default site view, to be provided to a group of users when information about the current context are not available.

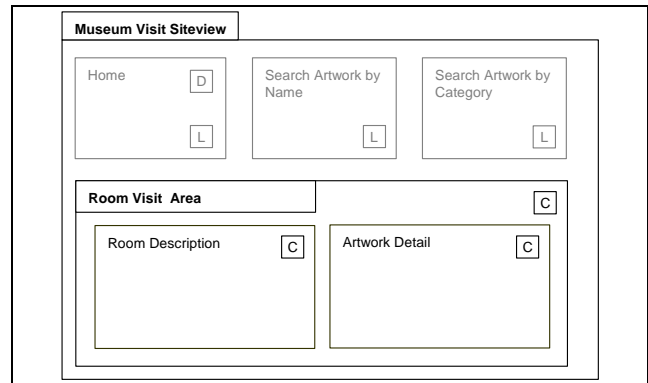
## 7. Extending the hypertext model for capturing context awareness

Besides introducing a data sub-schema for representing context, it is necessary to introduce some extensions also at hypertext level, for specifying context model operations and hypertexts adaptivity.

Our basic assumption is that context awareness is a property to be associated only to some container elements, i.e., pages, areas, and site views. The contained elements will inherit the adaptivity actions defined for their containers and, additionally, can be associated with more specific actions.

As represented in Figure 6, it is possible that within an application only some of the defined pages, areas, and site views need to be adaptive with respect to context. Some pages, for example “access” pages, as those reported in Figure 6 for invoking searching services, could not need adaptivity actions.

During requirements specification, the application designer will identify the context-aware elements and tag them with a C-label (standing for context-aware). The “C” label indicates that some adaptivity actions are specified for the marked element, and that, during the application execution, such actions must be evaluated and applied prior to element computation.



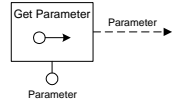
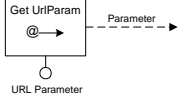
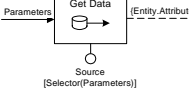
**Figure 6.** Graphical representation of context-aware areas and pages.

During the execution of the application, the process of context-based adaptivity is initiated by the request of a “C-labeled” page. This event triggers the acquisition of context information first, according to the three mechanisms already described in Section 4.2. If fresh context data are captured as session parameters, some operations for updating the context model within the application data source are performed. Finally, some operation chains allow retrieving possible data from the data source, or also computing new data, needed for adapting the page to the renovated context status. Only after such actions, the page computation starts.

While the user is still visiting the page, the context can change, for example because the user moves to a new location. In order to track the context status, and eventually undertake adaptivity actions, we associate some refresh properties to “C” pages, so as the associated adaptivity actions can be eventually repeated.

The central context-aware element is therefore the page. As shown in Figure 6, we however propose to define “C” areas as grouping facilities, that allow us to insulate some redundant adaptivity actions to be performed for every “C” page within the area, and represent them once at the area level. The adaptivity actions associated to areas are to be executed every time a “C” page within the area is accessed, and before executing the adaptivity actions specified at the page level. Typical actions to be specified at the area 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 context model updating. Such actions can be associated with the most external containers (site views or areas), and are inherited by all the internal elements (pages or areas).

<b>Get Unit</b>		It specifies the retrieval of the current value of a global parameter, previously set through a set unit.
<b>Get URL unit</b>		It specifies the retrieval of the current value of a context parameter appended to the page request.
<b>Get Data unit</b>		It specifies the retrieval of values extracted from the database according to the specified selector condition.

**Table 4.** *Get Parameter*, *Get URL-Parameter* and *Get Data* units.

- *Actions for hypertext adaptivity*, addressing the definition of rules for page and navigation customization. Such actions are generally associated with “C” pages.

The main profit of extending the “C” property to areas and site views is the reduction of specification redundancy. Therefore, in general we assume that each request of a “C” page causes first the computation of actions associated to its “C” containers (if any), starting from the most external one.

In the following sections, we will introduce some new hypertext constructs that allow us to express actions for context-model management and hypertext adaptivity.

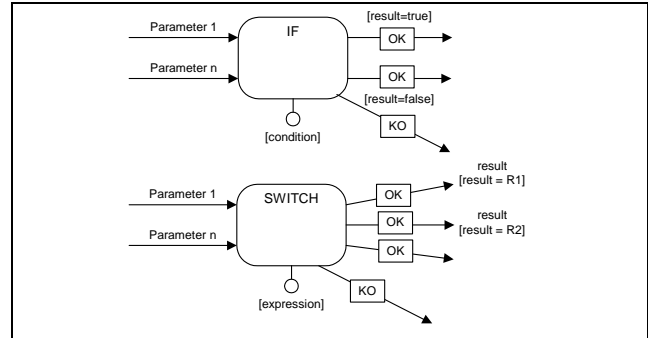
### 7.1 Units for accessing fresh context data

In order to specify actions for accessing fresh context data, according with the three access mechanisms illustrated in Section 4, we use three units able to access both sensed data and database objects. Table 4 presents their visual notation.

The *Get* unit is the one used in WebML for accessing global parameters, as already described in Section 2 and formally specified in [4].

The *Get URL* unit differs from the *Get* unit in that it refers to user-device generated parameters, appended to the request of the current page. WebML already supports the specification of *link parameters* and *global parameters* that at runtime correspond to values appended to the page URL. However, those parameters are server-generated, while through the *Get URL* unit we intend to specify parameters generated by client-side sensing mechanisms.

Finally, the *Get Data* unit is used for extracting values (both scalar values and values set) from the data source,



**Figure 7.** IF and SWITCH control primitives as WebML units.

according to a selector condition. It is similar to the data unit, with the only difference that the retrieved data are not published in a page, but just used as input to some successive operations. It is therefore a facility for accessing database information whenever no visualization is needed, e.g. in situations where certain data are just used to evaluate condition expressions.

### 7.2 Units for condition evaluation

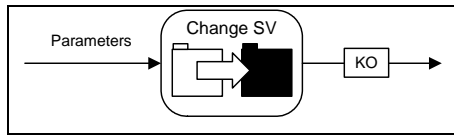
The execution of context model updating, as well as of hypertext adaptivity actions depends very often on the evaluation of some conditions. At this proposal, in order to specify test over certain conditions, we use two control constructs, the *If* and the *Switch* operation (see Figure 7), that have been recently proposed for extending WebML for modeling Workflow [1].

The two units have incoming links carrying possible parameters, and use such parameters for evaluating an expression. If the evaluation succeeds, one of the output *OK* links is followed, depending on the result of the evaluation. In particular, the *If* unit evaluates a Boolean expression, and provides two different *OK* links, one to be followed when the condition is true, the other when the condition is false. The *Switch* unit evaluates an expression, defined over the input parameters, and provides a set of *OK* links, to be followed depending on the matching between the results of the expression and the guard condition specified over the link.

If the expressions for the two units are not computable, the *KO*-link is followed. In addition, the switch unit presents one *OK*-link without guard condition; which is followed when the computation results do not match any of the guard conditions, but the calculation has been successful.

### 7.3 Unit for site view switching

In some cases, the change of context may ask for a change of site view. In order to specify this action, we have introduced the *ChangeSiteView* unit (see Figure 8 for its visual notation).



**Figure 8.** Change SiteView unit.

The unit takes in input the identifier of the new site view and of the specific target page within it. The input link also transports all the parameters required by the target page for reconstructing the current state, as well as those parameters representing selections operated by the user while interacting with the origin site view and that can be useful for the computation of the new page.

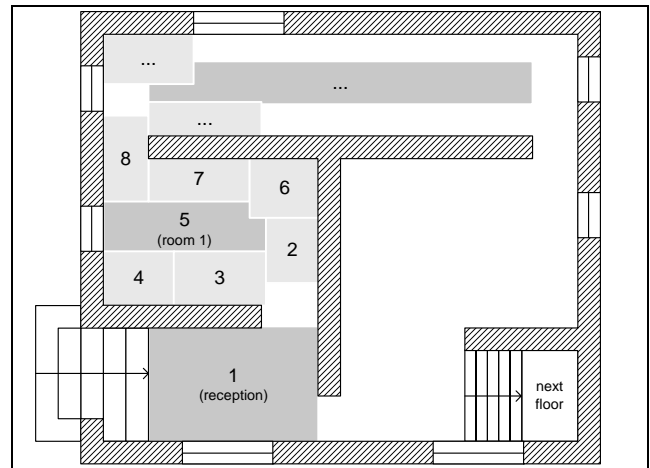
## 8. Case study: the museum guided tour

In this section, we show – by means of a simple example – how to integrate application data and context data for the provision of context-aware hypertexts.

Our scenario refers to an application assisting visitors of a museum. It can be seen as a new enlargement of an already existing Web application, publishing general information about the current and future expositions, its artworks and its artists, and providing the user with a reservation mechanism for the various guided tours through the museum. The desired extension is a guided indoor visit. It allows users to walk along the museum rooms, accessing the description of exposed artworks through a portable device, for example a PDA, able to access the wireless network inside the museum.

For location detection, we suppose the device being able to sense its position and adding the result as parameters to each page request. The device is able to gather the signals that the numerous transmitters installed above the exposed artworks emit, which allows locating the visitor's position inside the museum and delivering the associated contents. The positioning of the transmitters forms sensitive cells within the rooms (see Figure 9) that the receiver is able to identify and communicate back to the application via a URL-parameter that carries the unique identifier of the corresponding area. Whenever the visitor enters a new cell, the application shows the description of the corresponding artwork or artist or, when the user is in the center of the room, it shows a general description of that room.

The museum guided tour aims at exhibiting maximum of adaptability by automatically changing the content displayed on the PDA as a function of the position of the user; it can be considered as an extreme case of adaptability, whose usability needs to be thoroughly assessed.



**Figure 9.** Plot of the ground floor of the museum, showing the various sensitive areas that can be associated to the user's position.

### 8.1 Data modeling

As reported in Figure 10, the data schema related to the indoor visit application supports the delivery of data concerning artworks, artistic movements and artists' biographies.

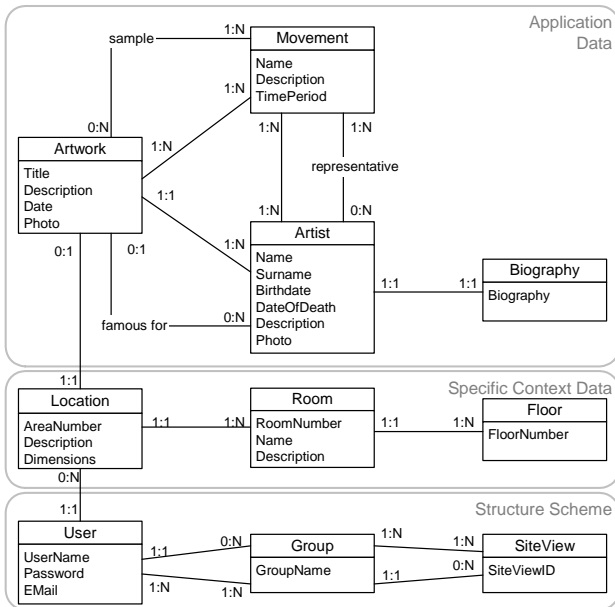
The bottom of the figure is about the *Context Model*. In this case, for sake of simplicity, beside entities representing the user model, only the *Location* entity is used for completing the context representation. Its attributes reflect the specific features of the museum infrastructure previously described. Its instances represent all the sensible areas for which the application is able to supply descriptions. In a given moment, a user can be associated to a specific location. If the user location changes, then the relationship between User and Location gets updated.

The connection point between context data and application data is the relationship between sensitive areas inside the rooms and the exposed artworks. The relationship expresses that each area exposes always exactly one artwork, while not every artwork is placed in an area inside the museum; the dataset may indeed contain artworks that are not currently exposed to visitors. Connected to the *Artwork* entity, we can then find the remaining application data, modeled by means of the entities *Artist* and *Movement*.

### 8.3 Hypertext modeling

Figure 11 reports a fragment of the hypertext schema for the museum application, which will allow us to show the new introduced extensions at work.





**Figure 10.** Entity-Relationship schema of the museum application, and identification of Application Data and Context Data.

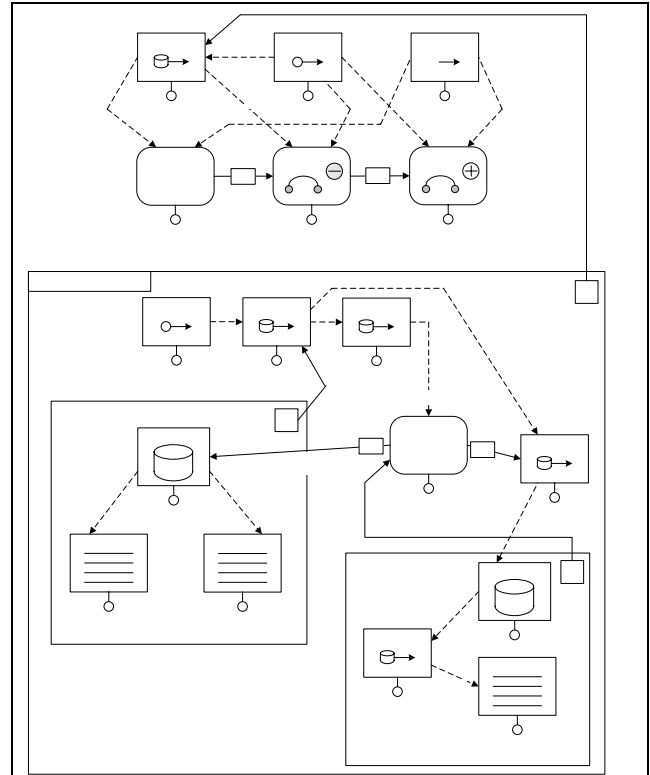
The hypertext represents a context-aware area (as represented by the C label at the right upper corner), which includes two context aware pages: one presenting details of artworks exposed in the current user location, the other showing details about the current room.

The “C” property associated to the area triggers an action chain, for the retrieval of fresh context parameters and the consequent context model updating, whenever one of its “C-labeled” pages is accessed. The chain starts with the retrieval of the current user identifier (*Get User* unit), then of the location currently associated to the user (*Get Location* unit), and finally of the (possibly new) user location sensed through the device (*Get URLParam* unit).

The user location stored in the data source and the new sensed location are used by the *If* unit, for evaluating if the two values are different. If yes, the context model needs to be updated. Therefore:

- A *Disconnect* operation deletes the current relationship between the user and the currently stored location;
- A *connect* operation creates the new relationship between the user and the location instance corresponding to the new sensed value.

As actions associated to the context-aware pages, the figure shows some operations for customizing the content of the current page to the current user location.



**Figure 11.** Hypertext fragment of the museum application.

The chain of get units, represented in the top of the figure, extracts from the database the artwork associated to the user’s current position and provides the following *If* unit with this value. The *If* unit checks whether there exists an artwork exposed within that position and, in case the artwork has been found, updates the data unit of the page *Artwork Details*. If there is no artwork associated to the current user position, it means that the user is located in the center of the current room. Therefore, the *If* unit triggers a page change by first extracting the respective room identifier from the database and then forwarding the user to the page *Room Details* by means of a contextual link carrying the retrieved identifier. This action implies the presentation of the page *Room Details* in place of the page *Artwork Details*, i.e., it represents an adaptivity action over navigation.

The page *Room Details* is aimed at presenting a description of the room, plus the index of its exposed artworks. Its adaptive actions are similar to those defined for the *Artwork Details* page, and cause a page updating with new contents, or a navigation towards the *Artwork Details* page.

## 9. Conclusions

This paper has presented some issues for the design of multi-channel context-aware applications, and has proposed some solutions for their conceptual modeling.

Several approaches have been proposed for the development of such class of applications (see [6] for a survey). However, since very few methods are model-based, often customization results into programming scripts buried within the application code.

The solution we have proposed is based on the adoption of the WebML conceptual model, and consists of the introduction of a new modeling dimension, for specifying the context model, as well as the required operations for context model management and hypertext customization. It offers the advantage, which is proper of conceptual modeling, to reason at a high level of abstractions, without being influenced too much by implementation issues. Also for this reason, we believe that, although proposed within WebML, our solutions can be easily adopted within other models or methods for Web applications development.

Our current work focuses on formally defining the semantics of the new introduced constructs, and on identifying possible changes to hypertext computation logic the extensions require for. We are also studying some synchronization issues that might arise in case of parallel sessions, through different devices, by the same user.

In the continuation of the MAIS project, which support this research, we aim at studying multi-modal applications, i.e., applications which take place by synchronizing the delivery of coordinated contents over two or more media; we will probably consider first visual and audio combinations. This research includes classical aspects of multi-media delivery (such as synchronization on various devices) but with the addition that each delivery can be considered as a Web application in its own rights, e.g., with browsable user interfaces (which, in the case of audio channels, may correspond to receiving explicit commands from the users and sending back vocal messages). Adaptation in this case may affect one or both of the channels (e.g., amplify one of them when the other one cannot be properly used due to changes to the context). The modeling will in this case require not only the adaptation of individual site views, but also the coordination among multiple site views.

On a separate ground, we will consider how the extensions which are proposed in this paper, and currently tested by "encoding" adaptability in a conventional Web application, can be implemented on

WebRatio, the current WebML case tool [9]; this will entail both a modification of the site designer component, to enable the model extensions, and of the automatic generation of the application from the specifications. Both changes are not particularly hard, due to the extensibility of the WebRatio architecture.

## 10. Acknowledgements

We are grateful to the WebML team for the useful discussions and the valuable suggestions.

This research work is funded by the MAIS (Multi-channel Adaptive Information Systems) FIRB project.

## 11. References

- [1] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu, Specification and Design of Workflow-Driven Hypertexts. *Journal of Web Engineering*, 1(2), April 2003.
- [2] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, I. Manolescu: "Model-driven Development of Web Services and Hypertext Applications", SCI2003, Orlando, Florida, July 2003
- [3] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, 2002.
- [4] S. Ceri, P. Fraternali, and M. Matera. Conceptual Modeling of Data-Intensive Web Applications. *IEEE Internet Computing*, 6(4), July-August 2002.
- [5] S. Ceri, P. Fraternali, and S. Paraboschi. Data-Driven One-To-One Web Site Generation for Data-Intensive Applications. VLDB '99, Edinburgh, UK, September 1999.
- [6] G. Kappel, B. Proll, W. Retschitzegger, W. Schwinger. Customization for Ubiquitous Web Applications – A Comparison of Approaches. *International Journal of Web Engineering and Technology*, January 2003.
- [7] A. Kobsa, J. Koenemann, W. Pohl: Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships. *The Knowledge Engineering Review*, 16(2), 2001.
- [8] D. Schwabe, R. Guimaraes, G. Rossi: Cohesive Design of Personalized Web Applications. *IEEE Internet Computing*, 6(2), 2002.
- [9] S. Ceri, P. Fraternali, R. Acerbis, A. Bongio, S. Butti, F. Ciapessoni, C. Conserva, R. Elli, C. Greppi, M. Tagliasacchi, G. Toffetti. Architectural Issues and Solutions in the Development of Data-Intensive Web Applications, CIDR2003, Asilomar, USA, January 2003.