Politecnico di Milano
Dipartimento di Elettronica e Informazione
Dottorato di Ricerca in Ingegneria dell'informazione

Model-Driven Design of Context-Aware Web Applications

Tesi di dottorato di:
Florian Daniel

Relatore:
Prof. Stefano Ceri
Correlatrice:
Prof. Maristella Matera
Tutore:
Prof. Letizia Tanca
Coordinatore del programma di dottorato:
Prof. Patrizio Colaneri

2007 - XIX

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32 I 20133 — Milano

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Dottorato di Ricerca in Ingegneria dell'informazione

Model-Driven Design of Context-Aware Web Applications

Doctoral Dissertation of:
Florian Daniel

Advisor:
Prof. Stefano Ceri

Coadvisor:
Prof. Maristella Matera

Tutor:
Prof. Letizia Tanca

Supervisor of the Doctoral Program:
Prof. Patrizio Colaneri

2007 - XIX edition

To my family and friends

Riassunto

L'evoluzione dell'Information Technology negli ultimi anni ha visto il World Wide Web trasformarsi da un media ipertestuale di sola lettura in una piattaforma matura per l'erogazione di applicazioni multi-canali e multi-servizi. Si è assistiti quindi all'evoluzione di semplici siti Web statici in vere e proprie applicazioni Web complesse, anche basate sull'uso intenso di dati. Per quanto riguarda lo sviluppo di applicazioni per il Web, questa evoluzione ha fatto nascere la necessità di metodi di sviluppo appropriati che siano capaci di far fronte alla complessità crescente ed alle peculiarità specifiche delle nuove generazioni di applicazioni Web. È il campo del Web Engineering che si propone a fornire risposte a queste nuove richieste, concentrando gli sforzi sullo sviluppo di metodologie e soluzioni sistematiche che permettano di raggiungere processi di sviluppo efficienti per le applicazioni Web moderne.

Con l'avvento dei nuovi dispositivi mobili, dotati di elevate capacità di calcolo, il Web riesce a raggiungere un numero sempre più crescente di utenti e a penetrare sempre di più nella nostra vita quotidiana. Si manifesta quindi sempre di più il bisogno di migliorare l'esperienza d'uso dell'utente finale, per esempio adattando l'applicazione alle sue preferenze o alle caratteristiche del dispositivo in uso. In questo ambito, la personalizzazione e, in generale, l'adattamento hanno già dimostrato i loro benefici sia per i fornitori di applicazioni sia per i consumatori di contenuti e/o servizi.

Analogamente, anche la context-awareness e meccanismi adattativi più flessibili stanno diventando sempre più un fattore vincente per migliorare sia l'efficacia sia l'efficienza delle applicazioni Web di oggi e soprattutto di domani. Con "context-awareness" si intende la capacità di tenere conto di qualsiasi proprietà o informazione che caratterizzi l'interazione dell'utente con l'applicazione, cioè il contesto, e di reagire a cambiamenti che tali proprietà o informazioni possono manifestare durante l'uso dell'applicazione. Adattamenti dell'applicazione non sono quindi più basati sulle sole preferenze dell'utente o sulle caratteristiche del dispositivo, ma più in generale su qualsiasi proprietà che caratterizzi il contesto dell'interazione. Adattamenti tipici di un'applicazione Web sono, per esempio, l'adattamento di contenuti o di link ipertestuali, l'esecuzione di

operazioni o servizi, o l'adattamento di proprietà di presentazione o di stile.

In linea con queste considerazioni, questa tesi si concentra sullo sviluppo di applicazioni Web context-aware e adattative. La risposta che questa tesi propone alla sfida posta al mondo del Web Engineering consiste in un metodo di sviluppo concettuale e model-driven per la progettazione di funzionalità context-aware e adattative nel Web. Il metodo proposto si basa sull'estensione di un linguaggio di modellazione concettuale che ha già affermato la sua validità, il Web Modeling Language (WebML), che fornisce anche uno strumento per la generazione automatica del codice dell'applicazione. Il modello di design proposto riflette una concettualizzazione di problemi e soluzioni relativi all'uso di funzioni context-aware e/o adattative nel dominio del Web e rappresenta una risposta approfondita ai principali requisiti che caratterizzano lo sviluppo di applicazioni Web context-aware.

Questa tesi fornisce uno dei primi approcci concettuali alla context-awareness e all'adattività nel campo del Web Engineering. Più precisamente, questa tesi rappresenta uno dei primi tentativi di allargare l'applicabilità di funzionalità adattative nel Web dai sistemi ipermediali adattativi ad applicazioni Web context-aware. Mentre i primi tipicamente sono basati su un profilo utente dinamicamente aggiornati in base a osservazioni della navigazione dell'utente, le ultime possono essere basate anche su un modello di contesto più complesso e funzionalità attive che siano attivate da questo contesto. Nonostante la ricerca descritta in questa tesi sia applicata al metodo di sviluppo WebML, la natura generale dei risultati ottenuti fa sì che questi contribuiscano anche all'avanzamento del campo del Web Engineering in generale.

La tesi parte con un'introduzione dei concetti di fondo che sono necessari per la comprensione delle idee sviluppate; si descrivono cioè le nozioni di context-awareness, adattabilità, adattività e il linguaggio di modellazione WebML. A partire da un confronto delle soluzioni adattative proposte in altri metodi di sviluppo per applicazioni Web, la tesi poi introduce gradualmente i nuovi concetti relativi all'adattività e alla context-awareness da applicare a WebML, descrive l'implementazione del modello esteso nell'ambito dello strumento CASE per lo sviluppo di applicazioni WebML, WebRatio, e discute un esempio di modellazione. La tesi poi mostra come i risultati ottenuti sono stati utilizzati in contesti diversi e come noi prevediamo che l'approccio proposto si possa sviluppare in futuro. Il capitolo conclusivo riassume il lavoro presentato, ne discute i punti di forza e i suoi limiti e anticipa possibili lavori futuri.

Abstract

The evolution of the Information Technology in the last years has seen the World Wide Web transforming from a read-only hypertext media into a full-fledged, multi-channel and multi-service application delivery platform. As a consequence, there has been an evolution from simple, static Web sites to complex, data-intensive Web applications. As for the development of such Web applications, the described evolution demands for appropriate development methods, able to cope with the growing complexity and the specific peculiarities of such new generations of Web applications. It is the field of Web Engineering that addresses this demand and that aims to develop systematic methodologies and solutions for an efficient development process for modern Web applications.

Also, with the advent of new and powerful mobile devices, the Web is addressing a continuously growing number of users and is more and more pervading our everyday life. In this regard, the need to improve the user's browsing experience, e.g., by adapting the application to user preferences and device characteristics, has become manifest. Personalization and adaptation to preferences and devices have already proved their benefits for both application providers and content or service consumers.

Similarly, context-awareness and more flexible adaptation mechanisms are increasingly becoming key factors to enhance both the effectiveness and the efficiency of the Web applications of today and especially of tomorrow. "Context-awareness" is intended as capability to take into account whichever properties or information that characterize the interaction with the application, i.e. the context, and to react to changes that such properties or information may experience during the use of the application. Reactions, i.e. application adaptations, are therefore not anymore based on the sole user preferences and device characteristics, but more in general on any property that characterizes the context of the interaction. Typical application adaptations in Web applications are, for example, the adaptation of contents or hyperlinks, the execution of operations or services, or the adaptation of presentation or style properties.

In line with these considerations, this dissertation puts its focus on the development of context-aware and adaptive Web applications. As

answer to the challenge faced by the Web Engineering field, this dissertation proposes a conceptual, model-driven method for the design of context-awareness and adaptivity in Web applications. The proposed method is achieved by extending an already established conceptual modeling language for Web application design, i.e. the Web Modeling Language (WebML), also providing for the automatic generation of the application code. The proposed design model reflects a conceptualization of problems and solutions deriving from the use of context-aware and/or adaptive features in the domain of the Web, thus representing a comprehensive instrument covering the main requirements in the design of context-aware Web applications.

This dissertation provides one of the first methodological approaches to context-awareness and adaptivity in the field of Web Engineering. More precisely, this dissertation is one of the first attempts to enlarge the applicability of adaptive application features in the Web from "adaptive hypermedia systems" to "context-aware Web applications". While the former typically are based on a user model that is dynamically updated based on the observation of the user's navigation actions, the latter may be based on a more complex context model and active, context-triggered application features. Although the research described in this dissertation is applied to the WebML method, its general nature also contributes to the advancement of the Web Engineering field in general.

The dissertation starts with an introduction to the background knowledge required for the comprehension of the outlined ideas, i.e. context-awareness, adaptability, adaptivity, and the Web Modeling Language (WebML). Inspired by a comparison of solutions provided by other Web modeling methods, the dissertation then gradually introduces the new concepts related to adaptivity and context-awareness into WebML, describes the implementation of the extended model in the context of the WebRatio CASE tool for the design of WebML applications, and discusses a modeling case study. Then, the dissertation shows how the achieved results have been exploited in different contexts and how we envision the work will evolve. The concluding chapter summarizes the presented work, discusses benefits and limitations, and outlines ongoing and future work.

Acknowledgements

Each time one concludes a period of his/her life, he/she feels the need to thank all those who contributed to or, however, positively influenced that experience. This dissertation is the result of one such period in my life, my Ph.D. studies at the Politecnico di Milano, and, hence, I indeed feel the need to thank a few people who I believe did contribute to make me feel comfortable during this experience.

There are for example Maristella Matera and Stefano Ceri, who are the actual reason for me sitting here today and writing about my Ph.D. If it was not for them, I probably wouldn't even have thought about doing a Ph.D. But they both insisted to the right degree, so I finally yielded to their idea and started this experience I now wouldn't like to miss anymore. Thank you for your advise and the trust you had in my work.

I would like to thank Giuseppe Pozzi for accompanying me in my first teaching experience and for the many interesting discussions during our lunches together.

I would like to thank Fabio Casati for hosting me as visiting researcher in HP Labs in Palo Alto, California. Although not directly related with this dissertation, the visit was an important part of my overall Ph.D. experience.

Then, there is some other people I would like to thank for their friendship: Federico Facca, Francesca Rizzo, Enrico Mussi, Pierluigi Plebani, Cinzia Cappiello, Stefano Modafferi, Danilo Ardagna, Marco Comuzzi, and Giovanni Toffetti Carughi. Federico and Francesca, it was always nice to work with you, and, Federico, it was great having you as office mate. Enrico, thanks for suffering so many times with me on the train when it was late, as usual. Gigi, thanks for all the fun. Cinzia, the next Stramilano is already waiting for us. Stefano, I'm sure we'll have lots of further discussions on our common hobby, beer brewing. Danilo and Marco, thanks for the amusing lunches together. Giovanni, thanks for the fun and all the useful discussions.

Of course, then there is my family, my parents and my brother, who always supported me in my decisions and believed in me. Thank you.

Also, I would like to thank Manu and her family for hosting me during the last years.

Finally, I would like to thank all the students who worked with me during this period and who actively contributed to the outcome of this dissertation and all the people I unwittingly forgot to mention here, but anyway contributed in some way to this dissertation.

Florian Daniel

Contents

1 Introduction

Current advances in communication and network technologies are changing the way people interact with Web applications. They provide users with different types of mobile devices for accessing – at any time, from anywhere, and with any media – services and contents customized to the users' preferences and usage environments. 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.

In the context of modern software systems, such as multi-channel Web applications, *adaptivity* is increasingly gaining momentum. Runtime adaptivity provides highly flexible and responsive means for the customization of contents and services with respect to the user's identity. Varying device characteristics in mobile and multi-channel computing environments can be adequately taken into account and leveraged by means of adaptive software designs, whose development is facilitated by the availability of standardized communication protocols (e.g. HTTP) and markup languages (e.g. HTML or WML), supported by most of today's mobile devices. Consequently, multi-channel deployment does not anymore require completely different, parallel design approaches and rather becomes a presentation[1] issue on top of unified engineering solutions. Then, in addition to user profile and device characteristics, adaptivity provides a way to also take into account a wide range of other properties describing the interaction between a user and a software system, thus paving the way for context-awareness and reactive behaviors.

Context-awareness is often seen as recently emerged research field in information technology and in particular in the domain of the Web. From the perspective of application front-end development it can however be interpreted as natural evolution of personalization[2] and adaptivity, ad-

[1] In this dissertation we use the term *presentation* with the meaning of "Web user interface".

[2] With *personalization* in this dissertation we intend the adaptation of the application to the user's identity and to explicitly provided preferences over contents and/or services.

dressing not only the user's identity and preferences, but also his/her usage environment. Personalization has already demonstrated its benefits for both users and content providers and has been commonly recognized as fundamental factor for augmenting the 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 combining personalization and adaptivity based on an application-specific set of properties (the context) that may affect the execution of the application.

Parallel to the evolution of the user's expectations, software systems are continuously getting more complex and difficult to maintain, partly also due to the previous, novel application requirements. Efficient abstraction mechanisms and design processes, such as those provided by *visual, model-driven design methods*, are thus becoming crucial. In this regard, the focus on essential design issues and the ease of reuse in model-driven design methods may significantly accelerate the overall design process. Starting from application models, code generation techniques may also provide for the automatic generation of application code or templates, thus assuring the fast production of consistent and high quality implementations. In addition, the high level of abstraction that characterizes model-driven design approaches and the ease of manipulation of high-level modeling constructs finally alleviate maintenance and evolution.

Given to such premises, this dissertation investigates the potential of adaptive or context-aware features in the design of Web applications and proposes a conceptual modeling approach addressing both adaptation and context-awareness. While adaptivity in general has been considered in the domain of the Web before (mainly in the context of adaptive hypermedia and e-learning systems), a review of the Web Engineering literature shows that context-awareness is an idea that has not yet been adequately addressed in this area. In this dissertation, both ideas are handled in a holistic fashion, and the resulting concepts yield an extension of the Web Modeling Language (WebML [1]), an already established model-based language for the design of data-intensive Web applications, accompanied by a development method and a CASE (Computer Aided Software Engineering) tool for automatic code generation [2]. Although the described conceptualization is formalized in the WebML design method, the introduced concepts have however a general validity and suggest a design methodology that may apply as well to other modeling methods.

Most conventional adaptive hypermedia systems mainly address the

problem of adapting the results of user-generated requests. Differently from such approaches, we also stress the importance of user-independent, *context-triggered* adaptation actions (i.e. the decision to apply an adaptation may derive from the user's interactions, but it is taken autonomously by the context). This aspect leads us to model context as a "first class" *actor* operating independently from users on the same hypertext the users navigate and represents a distinguishing feature of the presented work. This can be seen as a key to the interpretation of this dissertation with respect to other Web modeling methods.

1.1 Motivating Adaptivity in the Web

Adaptivity in general presents several potentialities of use, whose realization in the Web is nowadays more and more facilitated by the evolution of the respective Web technologies.

1.1.1 Use of Adaptivity

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

- *Adaptive personalization.* User profile attributes for personalization purposes may present different levels of variability in time. Profile properties may be static in nature (e.g. the name of a user), slowly changing (e.g. profile data derived from a user's browsing behavior) or even fast changing (e.g. the pulse frequency of a patient). Adaptive personalization mechanisms that take into account such profile peculiarities could allow systems to go beyond the common and static tailoring or services and contents.

- *Interaction-enabling functionalities.* Context could as well consider handicaps or physical disabilities of users, such as vision problems, blindness or paralysis, to adapt the application accordingly and to provide alternative and better suited interaction mechanisms and modalities. Adaptivity could thus provide functionalities enabling handicapped users to properly interact with applications, thus fostering the accessibility of applications.

- *Effective content delivery.* In general, whatever context data may be leveraged to provide appropriate contents and program features at the right time, priority, and emphasis. For example, specifically targeted special offers could be advertised and directed more effectively, presentation properties could be adapted to varying luminosity conditions for better readability, etc. Adaptive or context-aware extensions could thus enhance the overall effectiveness of applications by adapting individual application elements to varying users or usages of the application.

- *Delivery of context as content.* Applications may depend intrinsically and in a structural manner from context data. Location-aware applications, such as city map services or navigation systems, treat position data as core contents of the application and adapt them, supported by proper localization mechanisms. To such kind of applications, the use of context data represents a functional requirement, rather than an optional feature.

- *Exception handling.* Critical events during the execution of a software system may raise exceptions and require prompt reactions being performed. Process-based or workflow-driven applications, for example, represent a typical class of applications that constantly have to cope with exceptional situations in order to guarantee the consistent termination of a running process. Here, adaptive or context-aware mechanisms could be leveraged to capture respective application events and to enact the pieces of application logic that are necessary to handle the exceptional situation.

- *Production and control systems.* Critical production or control systems may require, for example, highly specific sensing and alerting mechanisms to prevent production losses or product quality degradations. Context-awareness could facilitate the timeliness of reactions and the efficient handling of dangerous situations, but also proactive maintenance approaches, such as those adopted in a steadily growing number of hardware/software systems, may be achieved.

- *Self-healing software systems.* Autonomic or self-healing software systems elevate the idea of proactive maintenance from hardware to software systems and aim at the creation of computing systems that are able to configure, tune, and even repair themselves. Proactive and adaptive capabilities in this context are an essential feature.

1.1.2 Adaptivity in the Web

Due to the current lack of appropriate technologies, concepts and Web design support, most of the previous application scenarios have actually not been considered suited to the domain of the Web or, in general, to be implemented with standard Web technologies.

Some recent initiatives to empower Web technologies, however, seem to be promising for extending the previous scenarios also to the Web. Traditional Web technologies (both hardware and software) are indeed continuously being evolved and extended, and considerable efforts are being invested to solve existing inadequacies. For example, think about the recent research efforts in the field of Web services that have produced numerous extensions (the often so-called WS-* specifications) to the original version of the service-oriented architecture and its communication protocols.

The effect of the ongoing research and evolution is that support for a multitude of non-functional requirements is being developed, whose initially inadequate coverage prevented the adoption of Web technologies in the above domains. For example:

- Communication protocols are becoming more and more reliable, thus providing room for the deployment on the Web of complex and critical applications, such as those related to production and control systems. The *reliability* of communication protocols has in fact been considerably enhanced along both the software and the hardware dimension. The introduction of reliable messaging techniques (e.g. digital certificates or the WS-Reliability specification) provides for trustworthy communications on top of standard Web protocols, such as HTTP or SOAP. The success of fiber optics, as an example of hardware evolution, has allowed the Ethernet protocol (typically used in the Web) even to enter industrial production environments, where the high electromagnetic interferences that may exist in the presence of high-voltage machineries practically prohibited the use of conventional, unreliable network technologies.

- On the other hand, the introduction of novel networking technologies, such as ADSL (Asynchronous Digital Subscriber Line) or fiber optics for home and office users and WiFi and 3rd generation mobile telephony technologies (e.g. UMTS, GPRS, EDGE) for mobile users, facilitates the continuously growing *availability* of Web applications or services.

- This evolution finally results into a higher technology *acceptance*

by end users and a higher *pervasiveness* of a great variety of applications and services. Such a widening of the audience, however, demands for efficient methods to address the variability of user characteristics and contexts of use.

- In addition, Web applications have already proved their flexible *scalability* (it suffices to think about certain portal applications that serve millions of users every day), facilitated *maintainability* and high *cost efficiency*. These three properties promote the Web as a cost-justifying and efficient deployment platform for a multitude of application domains.

Based on this perspective as well as on our personal assessment, we believe that the recent technology advances, complemented with appropriate development methods for Web applications, provide an adequate framework to enable the design of adaptive or context-aware Web applications. Technological advances indeed facilitate the development of adaptive Web applications, however, in general adaptivity needs to be considered more a development issue, rather than a mere functional/technological problem. In line with these considerations, this dissertation aims to provide a solid development method and a modeling instrument for context-aware Web applications, and to show the applicability of Web technologies to the aforementioned (and other) application domains.

1.2 Focus of the Dissertation

Instead of concentrating too much on technological issues, this dissertation, hence, aims to develop context-awareness for Web applications from a conceptual point of view – however, always with implementability in mind. For this purpose, in this work we extend the well known conceptual modeling language WebML (*Web Modeling Language* [1]) with a new modeling logic and with new design primitives. The extension of the modeling language also entails an extension of the overall design method of WebML applications, which allows us to contextualize the focus of this dissertation, as graphically summarized in Figure 1.1.

The WebML approach to the development of Web applications consists of different phases. Inspired by Boehm's spiral model [3] and in line with modern methods for Web and software applications development [4, 5, 6], the WebML process is applied in an iterative and incremental manner, in which the various phases are repeated and refined until results meet the application requirements. Such an iterative and incremental life cycle

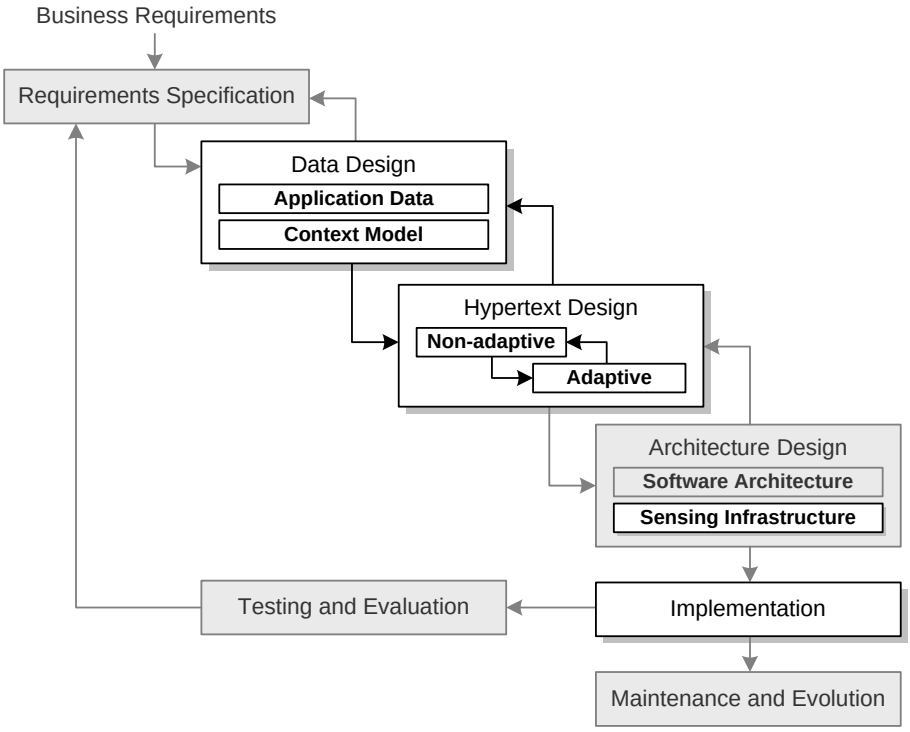


Figure 1.1: Phases in the WebML development process for context-aware Web applications. The steps addressed in this work are highlighted in white.

appears particularly appropriate for the Web context, where applications must be deployed quickly (in "Internet time") and requirements are likely to change during development.

The introduction of context-awareness and/or adaptivity into WebML needs to take into account new development tasks deriving from the use of context in the application design (the ones highlighted in white in Figure 1.1). Out of the entire process, the "upper" phases of analysis and conceptual modeling (i.e. data and hypertext design) are those that are most influenced by the adoption of a conceptual model. The more implementation-related phases can be automated to a high degree in the case of standard WebML applications, while, due to the peculiarities of each context-aware system (e.g. of the sensing infrastructure), in the case of context-aware Web applications the same degree of automation cannot be supported anymore, in general.

The most salient aspects of the extended development process can be summarized as follows:

1. *Requirements Specification.* Requirements specification focuses on collecting information about the application domain and the expected functions and on specifying them through easy-to-understand descriptions. The input to this activity is the set of business requirements that motivate the application development and, in case of context-aware applications, the set of desired adaptive behaviors with their triggering context properties. This task is not further elaborated in this dissertation.

2. *Data Design.* During this phase, the data expert organizes the main information objects identified during requirements specification into a comprehensive and coherent conceptual data model. Data modeling is a well-established discipline.

   However, data modeling for context-aware Web applications has a special flavor, due to the role assigned to the context model, and it is therefore one of the cornerstones of this work. In the context of WebML, explicitly representing context properties in the application's data source allows designers to express many useful adaptation behaviors declaratively in the hypertext specification.

3. *Hypertext Design.* Hypertext design is the activity that transforms the functional requirements identified during requirements specification into one or more site views (hypertext structures), embodying the needed information delivery and data manipulation services. Hypertext design operates at the conceptual level and exploits the WebML hypertext model that lets the hypertext architect specify how units, defined over data objects, are composed into pages, and how units and pages are connected by links to form hypertexts.

   According to the model-driven paradigm of WebML, also context-awareness and adaptivity can be modeled visually, instead of being buried in the source code of the application. This, however, requires the introduction of new modeling primitives and concepts, which are the main focus of the work presented in this dissertation.

4. *Architecture Design.* Architecture design concentrates on the definitions of the hardware, network, and software components by means of which the application delivers its services to the users. The inputs of architecture design are the non-functional requirements and constraints identified during the requirements specification, among them also requirements deriving from the use of context.

More precisely, in case of context-aware applications, architecture design also consists of the design of a context *sensing infrastructure*, tailored to the application requirements. Although we will discuss an example sensing infrastructure in our case study, context sensing is out of the scope of this work and therefore not further elaborated.

5. *Implementation.* Implementation is the activity of producing the software modules necessary to transform the data and hypertext design into an application running on top of the selected architecture. Hence, the inputs of this phase are the data model and the well formalized WebML schemas, one for each site view.

   The implementation (and architecture design) of WebML applications is largely assisted by the WebRatio development environment [2]. Starting from the extensions applied to the modeling language, we will show how the WebRatio environment can be extended to cope with the new requirements introduced into the implementation process by the use of context-aware application features. More precisely, we will discuss the development of two different prototype implementations: the first prototype shows how a subset of the context-aware features introduced in this dissertation can be achieved without touching the WebRatio environment; the second prototype, instead, shows how the new features can be implemented inside the WebRatio modeling tool. The two prototypes thus serve to prove the overall viability of context-awareness in Web applications and to show the implementability of the model-driven approach.

6. *Testing & Evaluation* and *Maintenance & Evolution.* Testing and evaluation is the activity of verifying the conformance of the implemented application to the functional and non-functional requirements, while maintenance and evolution represents the process that affects the application after the application has been deployed. These two phases are not further deepened in this dissertation.

1.3 Objectives

In line with the previous considerations, the objectives of the research described in this dissertation can be summarized as follows:

- Promote the use of *Web applications* to satisfy novel application requirements, i.e. adaptivity and context-awareness, thus widening

the applicability of Web applications and paving the road for future evolutions of the Web paradigm.

- Demonstrate the *feasibility* of context-awareness in Web applications that leverage existing technologies (both hardware and software) through a prototype application.
- Study domain-specific peculiarities and functional and non-functional *requirements* of context-awareness and adaptivity in Web applications.
- Provide meaningful *abstractions* and efficient *development support* for the design of applications that adapt to the context of use.
- Develop a *model-driven approach* to the design of context-aware Web applications as extension of the conceptual modeling method WebML, also enabling the automatic generation of the application code.
- Show how the *extensibility* of the proposed approach can be successfully leveraged to address some advanced adaptivity features.
- Discuss *benefits* and *limits* of the particular solution proposed in light of related works.

1.4 Definitions

Throughout this dissertation we will make use of a set of recurrent terms to discuss related work and to describe novel concepts. Newly introduced terms or names are always defined at their first appearance. In this section we however provide those definitions which we regard as fundamental for the comprehension of the presented work.

We start with the definition of *context* and *adaptation*, which are two of the main concepts this dissertation concentrates on. Our definitions have been inspired by the work of Dey and Abowd [7], further discussed in Section 2.1:

Definition 1 (Context) *Context is 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.*

Note that this definition does not take into account only interaction and environment properties, but it also includes properties of the user and the software system themselves.

Definition 2 (Adaptation) *Adaptation of an application to a particular context is the activity or process that the application needs to perform in order to satisfy the requirements posed by the context.*

According to this definition, adaptation to context can be achieved in a variety of forms. In the context of software systems in the Web, adaptation typically implies the re-computation of a page, the execution of software handlers, the updating of data, etc.

Starting from the previous definitions, we are now able to define the concept of *context-aware application*:

Definition 3 (Context-Aware Application) *A context-aware application is an application that uses context to deliver services or contents, that adapts to context, or that does both.*

Context-awareness in software applications therefore implies the capability to use context data to deliver contents, or to perform operations and/or to autonomously take decisions or enact operations. In this work we particularly stress the independent nature of context, which may change and evolve completely independently from the user's interaction with the software system. To underline this peculiarity, in this dissertation we will use the term *active context-awareness* to better convey the fact that context-awareness – at least in our interpretation – demands for active mechanisms that are capable of performing adaptations independently from the user interaction (ideally by means of a push mechanism). Current adaptive Web systems, for instance, typically adapt pages only in response to user-initiated page requests (pull mechanism).

Definition 4 (Active Context-Awareness) *Active context-awareness means that adaptations to new context conditions are performed automatically, at any time during application execution, and independently from possible user interactions.*

User-independent, automatic adaptations (i.e. active context-awareness) requires the application to continuously observe the state of context data, in order to be able to decide whether adaptations are to be performed or not. In this dissertation, we will call this activity *context monitoring*, to emphasize the difference of this approach from the one followed by adaptive hypermedia systems. In fact, in adaptive hypermedia systems there is typically no context monitoring, as the only observable behavior is the browsing of the user.

However, in addition to the previous definitions, in adaptive hypermedia systems two further adaptation-specific terms have been commonly

accepted [8]: *adaptability* and *adaptivity* (as well as their adjective counterparts *adaptable* and *adaptive*, used to characterize applications). Due to the introduction of context as adaptation trigger, in this work we provide two slightly extended definitions[3]:

Definition 5 (Adaptability) *Adaptability is the ability to adapt an application to device capabilities and/or user preferences prior to the execution of the application.*

Definition 6 (Adaptivity) *Adaptivity is the ability of an application to adapt to varying context conditions during the execution of the application.*

As such, adaptability is rather a (design time) property to be associated to the development method or tools that are used to design the application and to the technologies that are adopted to implement the application, and less to the application itself. Adaptivity, instead, refers to the application's capability to adapt during application execution (runtime), thus after the deployment of the application. Accordingly, adaptability is considered to be *static*, i.e. affected application properties are fixed at design/deployment time, while adaptivity is *dynamic*, i.e. affected application properties may change even after the deployment of the application. Dynamic adaptivity is typically harder to design than static adaptability, because of its tight integration into the application's functionality specification.

1.5 Structure of the Dissertation

This dissertation is structured as follows:

Chapter 2 introduces the reader to context-awareness and to context modeling, starting from a historical perspective. The chapter then provides another historical perspective, this time over Web applications and their evolution since the emergence of the World Wide Web. The two aspects, i.e. context-awareness and Web applications, are then joined by discussing how current conceptual modeling approaches for the design of Web applications support context-awareness and/or adaptivity in the design process.

[3] In particular, our definition of adaptivity is more general compared to the one given in [8], since adaptivity is not anymore triggered by the sole browsing behavior of a user stored in a user model, but instead by any context property stored in a context model.

Given the conceptual modeling approach followed throughout this dissertation, Chapter 3 introduces the Web Modeling Language (WebML), which will be used and extended to exemplify the new ideas to be applied to the domain of the Web. The chapter describes the basic modeling primitives and the WebML design process, so as to allow the reader to get familiar with the model-specific logic and to facilitate the comprehension of the following discussion. The reader already familiar with WebML can easily skip this chapter.

After the introduction of WebML, Chapter 4 applies the ideas from the fields of context-awareness and adaptivity to the WebML language. This chapter results into an extension of the language that enables Web developers to tackle generic, adaptive application features at a conceptual level, also paving the road for the automatic generation of the application code. This chapter is the core contribution of this dissertation, around which the remaining chapters are built.

Chapter 5, for example, discusses the automatic code generation process for adaptive Web applications, starting from the extended WebML schemas. The chapter provides some insights into the development of two prototypes that have been developed in the context of the Italian research project MAIS (Multichannel Adaptive Information Systems) and finally have led to the extension of the WebRatio CASE tool for the computer-assisted development of WebML applications. The chapter also describes the development of a proper client-server module to be added to adaptive Web pages, so as to enable active context-monitoring in the background.

Chapter 6 exemplifies some modeling peculiarities by describing a context-aware demonstration application providing location-aware information inside the Politecnico university campus. Special focus is put to the different modeling requirements of volatile and persistent context parameters in the extended WebML hypertext schemas.

Chapter 7 describes three correlated research works that have been developed starting from the context-aware extension of the WebML language described in the previous chapters. More precisely, first the chapter describes how the results achieved so far have been exploited in the context of the MAIS project in collaboration with other research groups. Then, the chapter describes a possible extension of the described approach that enables the capturing of composite user events (i.e. entire browsing behaviors) to trigger adaptivity rules. Finally, the chapter provides insight into a recent extension toward the runtime management of adaptivity features by means of event-condition-action rules.

Finally, in Chapter 8 we draw our conclusions of the dissertation, discuss the results obtained in the context of the Ph.D. program that

have led to this dissertation, and provide an outlook over ongoing and future work.

2 Context-Awareness and the Web

Since its emergence in the early nineties, where the World Wide Web imposed itself in the public as facility to browse documents and data in form of textual, static resources (the *hypertexts*), the Web has been evolving continuously. The use of additional media types has transformed hypertext into *hypermedia*, and the traditional one-way consumption paradigm has evolved to a full-fledged, two-way interaction. Context-awareness, on the other hand, has mostly been considered a separated research field. In fact, initially context-awareness was studied by people that typically were different from the people that worked on hypertext and Web technologies, i.e. there were separate research communities. Only recently, context-awareness, adaptivity, and personalization have been applied also to the domain of the Web. An analysis of the latest works published at international conferences like *Context*[1], *Adaptive Hypermedia*[2], or *User Modeling*[3], which initially were distinguished by different research focuses, confirms this trend.

In this chapter we discuss context-awareness and adaptive Web applications and show to what extent the two areas have merged so far. Special focus is put on conceptual modeling and model-driven approaches for the design of Web applications.

2.1 Context-Awareness and its Origins

In 1991 Weiser [9] published his revolutionary vision of ergonomic software and computer systems and stressed the term *ubiquitous computing*, which immediately became a commonly acknowledged idea and research field. Still today, researchers in the field of ubiquitous computing invest considerable efforts into the development of novel technologies and interaction paradigms with the aim of realizing Weiser's vision of computing

[1] http://www.context-05.org
[2] http://www.ah2006.org
[3] http://gate.ac.uk/conferences/um2005

systems and technologies that are no longer perceived by the user:

> "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." [9]

This relatively vague idea did not yet provide any specific solution, nor did there exist any technologies that could meet the requirements of the idea, but exactly this reason inspired a huge number of researchers. Very likely, also the research in the field of context-awareness, emerging in the early nineties, was inspired by Weiser's vision. Indeed, in order for a system to become part of the natural human environment, taking into account the context of the interaction and the execution environment seems to be the most basic, mandatory requirement posed to the system. The system, hence, needs to adapt to the user and to the environment – and not vice versa –, and this adaptation needs to be performed in a manner that is as transparent as possible to the user. Context-awareness thus seems to be one of the main ingredients to realize the idea of ubiquitous computing.

Schilit and Theimer [10] were the first to introduce the term *context-aware* in 1994. They referred to context as to location, identities of nearby people or objects, and changes to those people or objects. Dey and Abowd [7] describe context as the user's emotional state, focus of attention, location, and orientation, as well as the date and time of the interaction between the user and the system, and the objects and people in the user's surroundings.

These two interpretations of context provide a descriptive definition, which makes use of specific *examples* to describe the nature of context. Other definitions make use of *synonyms* to depict the concept of context; representative synonyms are for example: environment, situation, or surroundings. In [11] Dey and Abowd try to provide an *operational* definition that focuses on the main aspects of context: context refers to where you are, who you are with, and what resources are nearby.

As these attempts to define context show, due to the high dependency of context from the actual purpose of the application and from the specific application domain or user community, it is very difficult – if not impossible – to provide a universal definition of context that applies to all kinds of applications. Summarizing, we can say that context in context-aware applications describes three interrelated environments:

- *User environment*: it provides context properties that are related to the individual user of the system, e.g. identity, location, and social situation.

- *Physical environment*: it provides context properties that describe the space or surroundings in which the user is located, e.g. lighting condition or noise level.

- *Computing environment*: it provides context properties that describe the state or health of the resources (both hardware and software) that compose the context-aware system, e.g. number of available processors, devices accessible for user input and visualization, network capacity, or connectivity.

It is worth noting that neither the previous definitions nor our definition in Section 1.4 imply any restriction on how context data are acquired. Context data acquisition may occur in two distinct fashions, depending on the level of user involvement: implicit or explicit. *Implicit* acquisition means that context data is acquired without the intervention or (sometimes) knowledge of the user; this is for example the case of applications that automatically acquire location data via a GPS device or that dynamically compute user profile data based on the user's browsing behavior. *Explicit* acquisition, on the other hand, implies the involvement of the user in the acquisition process; this is mainly the case of applications that require the user to register and fill a proper user profile with personal data.

Independently from how context data are acquired, context data in context-aware applications is leveraged in two main fashions, i.e. *using* context and *adapting* to context, to provide support for the following typical application features [11]:

- *presentation* of information and services to users;

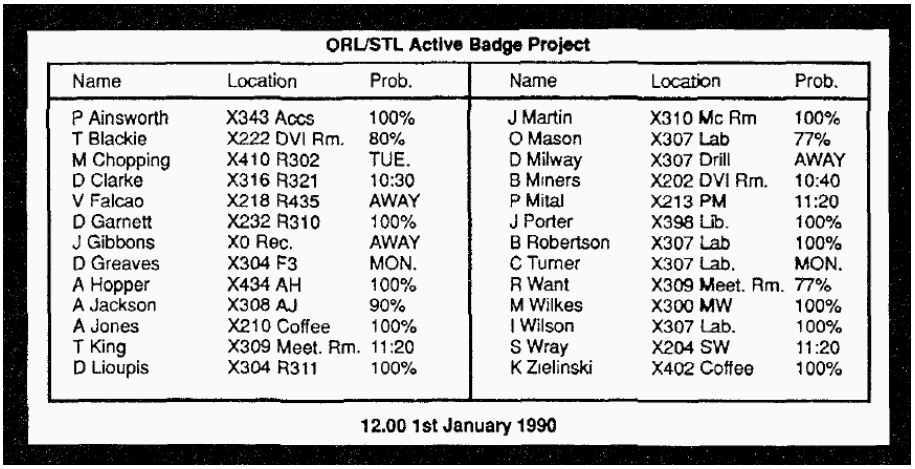
- automatic *execution* of services;

- *storing* of context for later retrieval.

Depending on the particular requirements to be developed, an application may implement one or more of these features, hence providing for different levels of context-awareness.

2.1.1 Two Historical Examples of Context-Aware Applications

The following examples illustrate two historical approaches to context-awareness pertaining to two typical application classes, i.e. office or meeting tools and (tourist) guides.



ORL/STL Active Badge Project

Name	Location	Prob.	Name	Location	Prob.
P Ainsworth	X343 Accs	100%	J Martin	X310 Mc Rm	100%
T Blackie	X222 DVI Rm.	80%	O Mason	X307 Lab	77%
M Chopping	X410 R302	TUE.	D Milway	X307 Drill	AWAY
D Clarke	X316 R321	10:30	B Miners	X202 DVI Rm.	10:40
V Falcao	X218 R435	AWAY	P Mital	X213 PM	11:20
D Garnett	X232 R310	100%	J Porter	X398 Lib.	100%
J Gibbons	X0 Rec.	AWAY	B Robertson	X307 Lab	100%
D Greaves	X304 F3	MON.	C Turner	X307 Lab.	MON.
A Hopper	X434 AH	100%	R Want	X309 Meet. Rm.	77%
A Jackson	X308 AJ	90%	M Wilkes	X300 MW	100%
A Jones	X210 Coffee	100%	I Wilson	X307 Lab.	100%
T King	X309 Meet. Rm.	11:20	S Wray	X204 SW	11:20
D Lioupis	X304 R311	100%	K Zielinski	X402 Coffee	100%

12.00 1st January 1990

Figure 2.1: A typical screenshot of the Active Badge system providing information about the location of office personnel inside the office building [12].

The Active Badge System

Schilit and Theimer first discussed the term *context-aware computing*, but it is commonly agreed that the first research investigation of context-aware computing was the Olivetti Active Badge work in 1992 [12], one year after Weiser's vision of "The Computer for the 21st Century".

With the Active Badge system, people could be located inside an office building, and phone calls could be directed to the phone closest to the called person. To associate locations to people, the office personnel wore badges that transmitted infrared signals (so-called active badges). A network of sensors placed around the office building picked up the signals, and a central location server polled the sensors. In this way, the telephone receptionist could easily find out in which room a person was located (see Figure 2.1) and forward the call to a phone of that room. According to the previous distinction of context properties, the Active Badge system adopted *user environment* context data; users are directly associated to a room.

In addition to the positioning of people, the system also supported commands to find out which other badges were in the immediate proximity to a named badge, to find out which badges were currently near to a specified location, to send out a notification as soon as a badge to be located was again traceable, and a possibility to obtain information about where the badge had been during a one-hour period.

The prototype was first installed in 1990 in Olivetti Research Center in

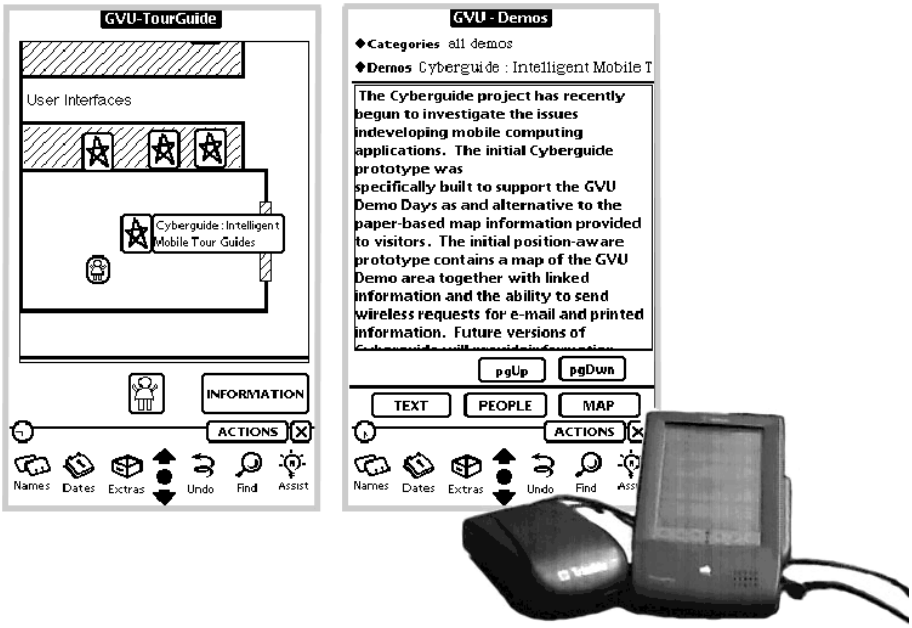


Figure 2.2: Screenshots of the indoor Cyberguide application and equipment for the outdoor Cyberguide application with GPS unit [13].

Cambridge, England. At the beginning, the personnel was worried about their privacy, but after a short experimentation phase many of them actually found the phone redirection service more useful than invasive. The system has later also been installed at other local sites and at Olivetti STL, Xerox EuroParc, MIT Media Lab and Xerox PARC.

Cyberguide

In the Georgia Tech Cyberguide project, mobile context-aware tour guide prototypes were built in the mid nineties [13]. The goal was to provide tourists with information based on their position and orientation. Initial prototypes of the Cyberguide were designed to assist visitors on a tour of the Graphics, Visualization and Usability Center during monthly open house sessions. The prototypes worked on an Apple MessagePad (with Newton OS) and used infrared beacons for user positioning. Information was initially stored on the MessagePad. With the help of Cyberguide, users could locate themselves in the building (i.e. view their current location) and identify the demonstrators in their surroundings on a map (see Figure 2.2). By selecting a demonstrator, it was possible to obtain

detailed information about its purpose.

Figure 2.2 shows also the equipment of the outdoor version of the guide, which has been implemented for guidance through the Georgia Tech Campus and for touring local establishments in Atlanta. Location sensing in the outdoor version was based on GPS coordinates.

Both Cyberguide systems (indoor and outdoor) make us of *user environment* context data. In the former case, users are associated to areas inside the building, in the latter case, the user position is described through GPS coordinates.

2.2 Using and Modeling Context

While the use of context in applications aims to ease users as much as possible in the interaction with an application, the use of context in the development of context-aware applications – in general – can not be regarded as easy. This reason has led to a significant amount of context abstraction efforts and development tools, out of which in this section we will concentrate on context modeling approaches and context management frameworks that aim to support the design of context-aware applications. In the following we narrow our focus on context modeling, but note that in general context modeling cannot be seen in complete isolation from user modeling.

2.2.1 Why is Context Difficult to Use?

There are several different reasons that make context cumbersome or even difficult to use. Dey and Abowd [7] identify for example the following characteristics:

- Context is acquired from *non-traditional* devices (i.e. not mouses and keyboards), with which we have only limited experience. Mobile devices, for instance, may acquire location information from the standard outdoor GPS system or from proprietary indoor positioning systems. In both cases, location sensing requires dedicated hardware (i.e. a GPS receiver and a proprietary location sensor) that must be programmed in order to interface the sensors with the application. Most of the times, this task implies the development of ad hoc solutions, as it is just impossible to cover all the possible application scenarios with off-the-shelf hardware and software (this is probably the real problem of context-aware applications).

- Context may be acquired from *multiple* distributed and heterogeneous sources. Tracking the location of users in an office may

require the system to gather information from multiple sensors throughout the office and, possibly, to calculate interpolations of position data, or similar.

- Context sensing technologies, such as video image processing, may introduce *uncertainty*. Context sensing for certain properties or in certain situations may only be able to provide results with an associated level of probability, e.g. a ranked list of candidate results. Uncertainty typically increases with the complexity of the sensing task; for example, detecting the presence of people in a room reliably may require the combination of results coming from the application of several techniques, such as image processing, audio processing, floor-embedded pressure sensors, etc.

- Context is *dynamic*. Applications must be able to adapt to constantly changing context conditions, and changes in the environment must be detected. One of the strongest requirements one can pose to a sensing infrastructure is for example the capability to sense in real time. A dynamic (and possibly historical) context model is needed to reflect the dynamic nature of context, comprising the context of use.

- Context must be *abstracted* to make sense to the application. GPS receivers for instance provide geographical coordinates (i.e. longitude and latitude). But a tour guide application for example – and in general any application providing context data to the user – would make better use of higher level information such as street or building names. Applications thus need the be able to perform context data transformations.

The first three of these context characteristics mainly imply technological or implementation issues, while the latter two also involve design issues at a higher level of abstraction: the dynamic nature of context asks for active or reactive application designs, and the need for a context abstraction to fit the application's own concepts asks for a suitable design of the context model underlying the application.

These two characteristics therefore heavily influence the design of the application and will thus be reconsidered in Chapter 4 when introducing our model-driven approach to the design of context-aware Web applications. While the former (i.e. dynamics of context) represents a rather intuitive problem, the latter needs some further elaboration to ease the comprehension of the discussion of existing context modeling approaches. The next section therefore provides some insight into context abstraction by dividing context properties into different abstraction levels.

2.2.2 Physical and Logical Context

Taking a more analytical view over context in general, allows one to identify a basic distinction between context properties that should help developers to understand how to derive and define a context model. According to the level of abstraction where context properties are situated, a context model can be separated into physical context and logical context [14, 15].

Physical Context

Physical context properties correspond to the real world properties that can directly be sensed by sensing hardware or software of a context-aware application. Physical context is therefore not modifiable, and it is outside of the control of the application. The sensed values just provide a manageable description of the environment and the application that can be taken in input by an application to provide services or contents or to adapt. Physical context properties are thus located at a very low level of abstraction, and they are continuously, dynamically updated, due to the fact that the environment and the application state itself may continuously change. Although the physical context is application independent, the application designer is able to specify which properties are of interest for a particular application, i.e. he specifies the physical context model.

Definition 7 (Physical Context) *Physical context is a virtual, manageable representation of real world (i.e. physical) context properties that can be sensed and made available to a software system through respective sensing devices.*

Physical context properties can thus be divided into three main sub-classes that collect similar characteristics; Kappel et al. [14] identify the following sub-classes:

- *Natural context.* The natural context comprises context properties like location, time, luminosity, humidity, etc.
- *Technical context.* The technical context includes data about the user agent (e.g. device and browser), the network (e.g. bandwidth, throughput) and the application itself.
- *Social context.* The social context holds data about the user for personalization purposes (e.g. the user's identity or preferences).

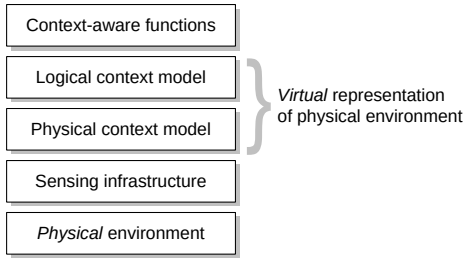


Figure 2.3: The building blocks of context-aware applications.

Context with the latest time stamp is called *current* physical context. A further distinction between *historical* physical context and *future* physical context (based on predictions/projections over current and/or historical context properties) may be necessary in function of the individual application requirements.

Logical Context Model

In contrast to physical context, logical context represents more abstract information about context. Logical context information is needed to enrich the semantics of physical context information (e.g. a cellID in mobile telephony), thus making it meaningful for application purposes (e.g. a street name). For each property that is part of the physical context model, the logical context model may provide appropriate logical information in order to augment the expressiveness of physical context data. Unlike physical context, logical context is under full control of the application designer; the logical context model can thus easily be extended by the designer to represent additional concepts or semantics necessary to capture the peculiarities of specific application domains.

Definition 8 (Logical Context) *Logical context refines physical context data by means of higher-level semantics, in order to augment the expressiveness of sensed data and translate them into application entities or concepts.*

Figure 2.3 provides a layered view over context-aware applications and shows the typical building blocks that characterize the software and hardware architecture of context-aware applications:

- *Physical environment*: it represents the interaction environment in which the user and the application operate.

- *Sensing infrastructure*: by means of suitable sensors, it gathers those properties from the physical environment that are required to support the context-aware features of the application.
- *Physical context model*: it collects sensed context properties in form of a manageable, virtual representation.
- *Logical context model*: it extends the physical context model with additional semantics.
- *Context-aware functions*: they use the underlying context model to provide context-aware application features.

Note how each building block in Figure 2.3 depends on the data and functionalities provided by the building block immediately below each block.

2.2.3 Context Modeling Approaches

The hight dependency of context from the specific domain in which a context-aware application is being developed and the intrinsic difficulties that arise when using context (see Subsection 2.2.1) make the specification of a universal context model an unrealistic task. Thus, instead of concentrating on the definition of a specific context model that could be applied in a variety of application domains, it is more interesting to study the different approaches that have been developed for the specification of the context models that underlie the applications. Rather than concentrating on a specific solution, in this section we therefore focus on the methodological aspect of context.

The initial momentum in the field of context-aware computing produced a variety of ad hoc solutions and specific applications (e.g. the Active Badge and Cyberguide systems described in Subsection 2.1.1), in which context data and the context model were hardwired into the application code. Reusability, extensibility or sharing of context was not an issue, and each application adopted proprietary solutions that were incompatible among each other. There was no conceptual distinction between application data and context data and, therefore, no independent management mechanism for context.

With the development of the Context Toolkit [16], Salber et al. made a first step toward reusable and configurable context management. The toolkit provided a set of abstractions (i.e. widgets, interpreters, and aggregators) as mediators between the environment and the application and a configurable execution framework for runtime support. The main

idea of the toolkit was to wrap the semantics and the logic of physical sensors by means of proper widgets (i.e. Java classes) that, once defined and added to the execution framework, could be reused in a variety of different and independent context-aware applications. Context abstraction was achieved by means of interpreters and aggregators that allowed to extend the semantics of widgets by means of other Java classes representing logical context information.

Although the Context Toolkit can be regarded as one of the first context conceptualization efforts, it however suffers of one main drawback that limits its extensibility and its widespread adoption: the context model must be hard-coded. That is, there is not separation between the description of context (i.e. the model) and the execution logic (i.e. the widgets) required for the management of the description. A context model in the Context Toolkit was thus rather specified in an *operational* fashion and less in a *descriptive* one, as context properties were tightly coupled to concepts from object-oriented software development.

In the following we disregard the operational aspect (i.e. the implementation) and concentrate on the descriptive definition of context by discussing the main conceptual modeling approaches that have emerged for the specification of context [17]: key-value models, markup schema models, graphical models, object-oriented models, logic-based models and ontology-based models.

Key-Value Models

Representing context properties as key-value pairs is the most simple modeling solution to specify context information. Already Schilit and Theimer [10] used key-value pairs to model context: they provided the value of context parameters to an application by means of environment variables.

The simplicity of this approach facilitates the management of small amounts of context data and limits the risk of errors. Key-value pairs, however, lack capabilities for sophisticated structuring of context data, which could enable efficient context retrieval and elaboration algorithms.

Markup Schema Models

Markup schema models are usually based on a serialization of a derivative of the Standard Generic Markup Language (SGML), the superclass of all markup languages such as the popular XML. Typical representatives of this kind of context modeling approach are profiles and extensions to the Composite Capabilities / Preferences Profile (CC/PP) [18] and User

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cscp="context-aware.org/CSCP/CSCPProfileSyntax#"
  xmlns:dev="context-aware.org/CSCP/DeviceProfileSyntax#"
  xmlns:net="context-aware.org/CSCP/NetworkProfileSyntax#"
  xmlns="context-aware.org/CSCP/SessionProfileSyntax#"
  <SessionProfile rdf:ID="Session">
    <cscp:defaults rdf:resource=
      "http://localContext/CSCPProfile/previous#Session"/>
    <device><dev:DeviceProfile>
      <dev:hardware><dev:Hardware>
        <dev:memory>9216</dev:memory>
      </dev:Hardware></dev:hardware></dev:DeviceProfile>
    </device>
  </SessionProfile>
</rdf:RDF>
```

Figure 2.4: CSCP profile example [17].

Agent Profile (UAProf [19]) standards, which have the expressiveness reachable by RDF/S and an XML serialization.

Common to all markup schema modeling approaches is a hierarchical data structure consisting of markup tags with attributes and content. In particular, the content of the markup tags is usually recursively defined by other markup tags. Figure 2.4, for example, shows a markup fragment taken from the Comprehensive Structured Context Profiles (CSCP) by Held et al. [20].

Markup schema models are a flexible means to represent context data. Schema definitions, such as DTD or XML Schema, can be used for structure validation and/or type checking, even in presence of partial context data. Schema definitions further represent a step toward formality, which may facilitate the interoperability between different models. De Virgilio and Torlone [21], for example, leverage this feature in their framework for representation and translation of context information, based on an intermediate representation format called General Profile Model (GPM).

Graphical Models

A well known general purpose modeling instrument is the Unified Modeling Language (UML) which has a strong graphical component (UML diagrams). Due to its generic structure, UML is also appropriate to model context.

Henricksen et al. [22] have extended the Object-Role Modeling (ORM) approach [23] into an intuitive and nicely designed graphics-oriented context modeling language. In ORM, the basic modeling concept is the fact, and modeling a domain in ORM means identifying appropriate fact types and the roles that entity types play in these. Henricksen et al. extended ORM to allow the definition of categorized fact types, taking into ac-

count their persistence and source in either a static (facts that remain unchanged as long as the entities they describe persist) or dynamic fashion. Depending on the source of the facts, dynamic facts are further distinguished into profiled, sensed, or derived types. Also, a new history fact type has been defined to cover the time aspect of context. The last extension of ORM for context modeling purposes are fact dependencies, which represent a special type of relationship between facts, where a change in one fact automatically leads to a change in another fact: the dependsOn relation. [17].

This kind of approach is further particularly suited to derive Entity-Relationship models from context models, which is very useful to translate the abstract context models into relational database structures for the use in context-aware applications. See Figure 2.5 for a description of the modeling notation.

Graphical models are powerful for the structuring of context information and, as the approach by Henricksen et al. [22] shows, they are a good starting point for the generation of application code from the model. Since graphical modeling languages are typically intended for human use only, this is a very useful feature.

Object-Oriented Models

Object-oriented context modeling approaches aim to employ the main benefits of the object-oriented paradigm – namely encapsulation and reusability – to cover parts of the problems arising from the dynamics of context. The details of context processing are encapsulated inside the object and, hence, hidden to other components or the developer. Access to contextual information is provided through specified interfaces only.

An example of modeling approach pertaining to this category is the Active Object Model of the GUIDE project [24]. All the details of data collection and fusion (e.g. the context-driven composition of HTML fragments) are encapsulated in the active objects and thus hidden to other components of the system. The approach has been primarily driven by the requirement of being able to manage a great variety of personal and environmental contextual information while maintaining scalability.

Object-oriented models are not considered being formal models, but a certain level of formality can be achieved through the use of well defined object interfaces.

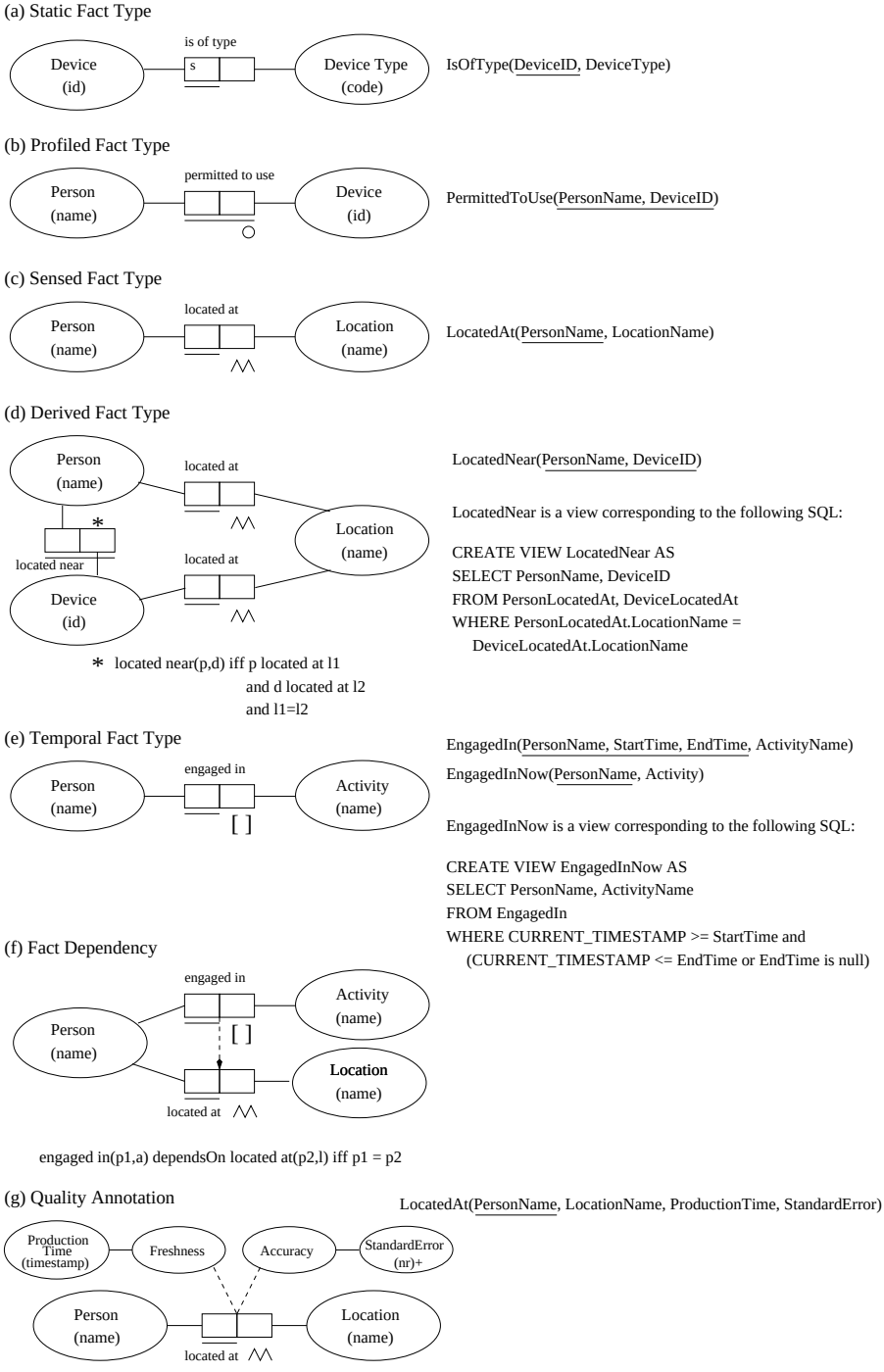


Figure 2.5: ORM extended for context modeling [22].

$$S_1 = [\dot{s} \mid \dot{s} \models\ll bird, \dot{a}, 1 \gg]$$
$$S_2 = [\dot{s} \mid \dot{s} \models\ll flies, \dot{a}, 1 \gg]$$
$$B \models\ll present, air, 1 \gg \wedge \ll penguin, \dot{a}, 0 \gg \wedge \ldots$$
$$C = S_1 \Rightarrow S_2 \mid B$$

Figure 2.6: An example of modeling with Extended Situation Theory [17].

Logic-Based Models

A logic defines the conditions on which a concluding expression or fact may be derived (a process known as reasoning or inferencing) from a set of other expressions or facts. To describe these conditions in a set of rules, a formal system is applied. In a logic-based context model, the context is defined as facts, expressions, and rules. Usually contextual information is added to, updated in, and deleted from a logic-based system in terms of facts, or inferred from the rules in the system, respectively.

One of the first logic-based context modeling approaches was published in 1993 by McCarthy [25]. He described context by means of abstract mathematical entities with properties as known from artificial intelligence, so as to facilitate the reasoning over context data. The main idea of the work is to bind the validity of a logical proposition to a specific context.

Figure 2.6, instead, depicts a context modeling example adopting the Extended Situation Theory by Akman and Surav [26]. Context is modeled with situation types, which are ordinary situations and thus first-class objects of situation theory. The variety of different contexts is addressed in form of rules and presuppositions related to a particular point of view. They represent the facts related to a particular context with parameter-free expressions supported by the situation type which corresponds to the context.

Common to all logic-based models is the high degree of formalization of the context model. This characteristic enables the use of reasoning techniques that could be adopted to support sophisticated dynamics in the context model or to deduce dependent context properties. On the other hand, however, the high level of formalization hinders the evaluation – and thus the use – of incomplete, partial, or ambiguous context information.

Ontology-Based Models

Ontologies are a universal instrument to specify concepts and relationships and, hence, to model information. Especially with the current interest in the Semantic Web and Semantic Web technologies (e.g. OWL and RDF(S)), ontologies are gaining momentum also in the domain of the Web.

A promising emerging context modeling approach based on ontologies is the CoBrA (Context Broker Architecture) system [27] by Chen et al. The CoBrA system uses a broker-based agent architecture to enable agents to acquire, reason about, and share context knowledge and to provide runtime support for context-aware systems. A key component of the proposed system is a context ontology defined in the Web Ontology Language (OWL), which contains the basic concepts of people, agents, places, and presentation events. The ontology describes the properties and relationships between these basic concepts, including relationships between places, roles associated with people in presentation events, and typical intentions and desires of speakers and audience members.

Due to their flexibility, extensibility, and increasing support for reasoning tools, ontologies are the most promising instruments for context modeling in context-aware applications. Also, the availability of visual ontology modeling tools and the growing adoption of standard ontology languages like OWL or RDF(S), accelerate the acceptance of and familiarity with ontology-based modeling techniques among developers.

2.3 Context and Web Applications

Context-awareness in general, until recently, has been mainly studied in the fields of ubiquitous, pervasive, wearable, or mobile computing. In the domain of the Web, so-called *adaptive hypermedia* systems [28] were the first to use a user's preferences, knowledge, and goals (thus *context*) during the interaction to adapt the hypertext to the needs of the user. As this dissertation concentrates on the design of context-aware Web applications, we briefly overview how context-aware or adaptive features have found there way into Web applications.

Looking back in history, the World Wide Web can be characterized by three generations of applications, differing in both the technology used and the services provided [14]:

- *Static Web pages* (1st generation): at the beginning, the Web was employed merely for simple read-only applications, presenting pieces of information to anonymous users whose number and

type was not necessarily predictable. Such usage affirmed the Web as one of the most powerful mass medias (if not the most powerful one) of today's world. Typically static Web pages are hand-made.

- *Dynamic Web pages* (2nd generation): later on, the Web was more and more used for increasingly complex applications, where huge amounts of change-intensive information were managed by underlying database systems, and Web pages were generated out of such data sources. The basis for promising application areas like e-commerce was built. Typically dynamic Web pages are automatically generated by filling page templates with application data or through XSLT transformations.

- *Ubiquitous access* (3rd generation): currently, we are facing a new generation of Web applications being characterized by the anytime/anywhere/any media paradigm, thus providing ubiquitous access to services, turning e-commerce into m-commerce. The prerequisite for realizing such services is awareness of context. Time-aware, location-aware, device-aware, and network-aware services are suitable to extend already existing customization concepts.

2.3.1 The Origins of Context-Awareness in the Web

Kappel et al. [14], for example, individuate the origins of customization with respect to ubiquitous or Web applications in two main streams, *personalization* and *mobile computing*.

Personalization

The notion of personalization represents a major challenge since the end user has been put in the middle of concern when developing interactive applications, which dates back to the early eighties. Personalization aims to provide users with an experience more tailored to their background knowledge and objectives. Personalization, however, is also beneficial to application developers, as it may enable a better re-use of application features and to target a broader range of users with one and the same application.

Adaptive Web applications aim at tailoring a system's interactive behavior and the content it visualizes to skills, tasks, and preferences of human users. The following classification describes four approaches where personalization has been used successfully:

- *Adaptive user interfaces*: these aim at tailoring the system's interactive behavior to skills, tasks, and preferences of human users

with special consideration of interface-related requirements.

- *Intelligent help and tutoring systems*: these systems adapt their explanations and teaching strategies to the individual needs of users in terms of their knowledge level and learning progress.
- *Information filtering and recommender systems*: they emphasize more on adapting the content of an application, and their main goal is to go through large volumes of dynamically generated textual information and present to the user those which are likely to satisfy his/her information requirements.
- *Adaptive hypermedia*: whereas the aforementioned approaches do not really consider the actual identity of users, and treat them all the same way, today Web applications are developed for an unpredictable number of anonymous users and work as if they were designed for each individual user. Each user can manage his own data objects, he has his own user profile and preferences over content and style properties.

Especially the customization concepts of this last class of applications are easily extensible, and context-aware mechanisms may adapt the application to varying context conditions during runtime.

Mobile Computing

The area of mobile computing can be seen as second major root of customization. In contrast to personalization, which has already a long tradition in application development, the research on mobile computing begun in the early nineties. According to [14], also mobile computing presents some main approaches:

- *Location-based services*: location information, which can be made available by mobile network providers or using technologies such as the Global Positioning System (GPS), is used for realizing various indoor or outdoor location-based services such as geographically targeted advertising, fleet management, traffic control, or emergency services. The two historical examples introduced earlier in this chapter are mainly location-based services.
- *Multi-channel Delivery*: this requires to consider the varying capabilities of devices in terms of hardware (e.g. display size and computational power) and software (e.g. operating system and Web browser) in order to allow a proper adaptation of the application's user interface and interaction behavior.

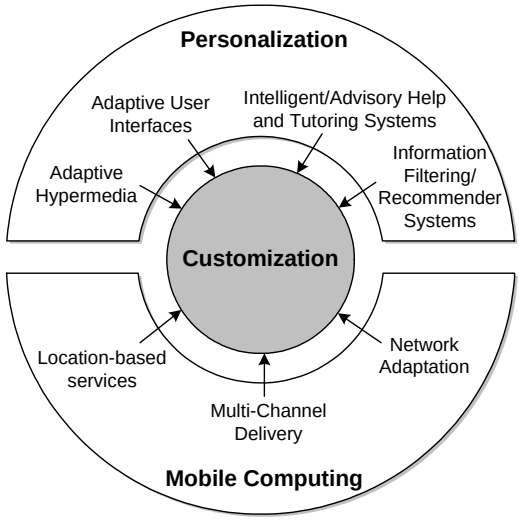


Figure 2.7: Summary of the approaches that influenced today's customization and adaptation approaches [14].

- *Network Adaptation*: in this respect, communication autonomy requires that the application properly adapts to sudden disconnections, which can be caused either voluntary, or happen against the will of the user because the battery runs empty or network connection is lost. Restrictions and variations in bandwidth may require to change either the content of the transmitted data or the methods used to send that data (e.g. by altering the underlying protocol).

Figure 2.7 summarizes the previous considerations on research areas and approaches that have led to today's customization and adaptation concepts.

2.3.2 Examples of Context-Aware or Adaptive Web Applications

A significant number of dedicated applicative solutions have been successfully developed [12, 29], and context abstraction efforts have produced proper platforms or frameworks for rapid prototyping and implementing context-aware software solutions [16]. Within the domain of the Web, so-called *adaptive hypermedia* systems [28] use a user's preferences, knowledge, and goals throughout an interaction to adapt the hypertext to the needs of the user. Recent research efforts also address the special needs of portable devices and mobile Web applications.

HyCon [30], 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. [31] represents a user modeling and adaptation tool originally developed in the e-learning domain. AHA! is delivered as Open Source software and provides a versatile adaptive hypermedia platform for the development of on-line courses, museum sites, encyclopedia, etc. According to a continuously updated user model, it allows customizing hypertext links (adaptive navigation) and contents (adaptive presentation).

Belotti et al. [32] 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 [33]. In [32] they describe their resulting general, context-aware content management system, which enables developers to seamlessly adapt content, view, structure, and presentation of Web applications to runtime context properties. Context affects the actual Web application indirectly by altering the state of the database and is not able to trigger autonomously application functionalities.

2.4 Model-Driven Design of Context-Aware or Adaptive Web Applications

At a more conceptual level, model-driven design methodologies aid developers in the design of complex Web information systems. However, despite the growing number of individual adaptive or context-aware Web applications, only few attempts exists that also aim at modeling adaptivity and/or context-awareness at the conceptual (i.e. design) level.

In this section we review some of the most prominent model-driven design methodologies by putting special attention to the different level of support that each of them provides for the specification of personalization, adaptability, adaptivity, and context-awareness. More precisely, in the following we review the following design methods: *Hera* [34],

OOHDM [35, 36], *OO-H* [37], *WSDM* [38], *UWE* [39], *OntoWebber* [40] and *SiteLang* [41].

We conclude this section with a comparison of the adaptive/context-aware features of the discussed approaches.

2.4.1 Hera

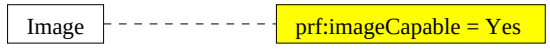
In [42] the authors show how the model-driven *Hera* design methodology [34] allows designers to specify the conditional inclusion or exclusion of hypertext fragments at content, navigation, and presentation level. Adaptation is achieved by means of so-called *appearance conditions*, attached to design artifacts and based on user profile and device capability information (i.e., CC/PP [18]). The described approach thus provides means for *static adaptation* (i.e. adaptability), where adaptation does not consider the user's browsing behavior.

Hera distinguishes three design models: *Conceptual Model*, *Application Model*, and *Presentation Model*. The Application Model extends the Conceptual Model with navigation primitives, and the Presentation Model enhances the Application Model with layout primitives. Concerning adaptation, each of the three Hera design models enables access to only those profile attributes that are meaningful in the context of the particular model.

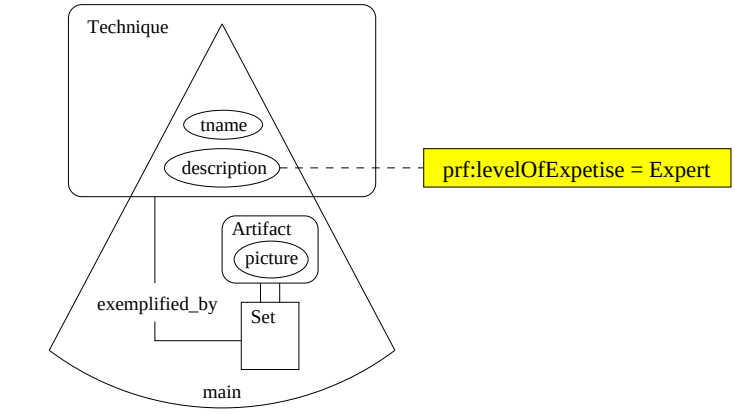
Figure 2.8 exemplifies the Hera approach to adaptation. Figure 2.8(a) shows the conditional inclusion/exclusion of concepts into/from the Conceptual Model; only if the condition attached to the concept is true, the concept will be part of the resulting Conceptual Model. Figure 2.8(b) shows how *slices* (i.e. meaningful data presentation units) in the Application Model can be suppressed by attaching proper appearance conditions to the respective modeling primitive; in the example, the textual description of the `Technique` concept will only be showed to users with a level of expertise equal to `Expert`. Finally, Figure 2.8(c) shows how so-called *regions* in the Presentation Model can be conditionally visualized or hidden; in the figure, based on the `prf:client` attribute of the profile, either `Region 3` or `Region 3a` (or none, if none of the conditions is true) will be rendered.

2.4.2 OOHDM

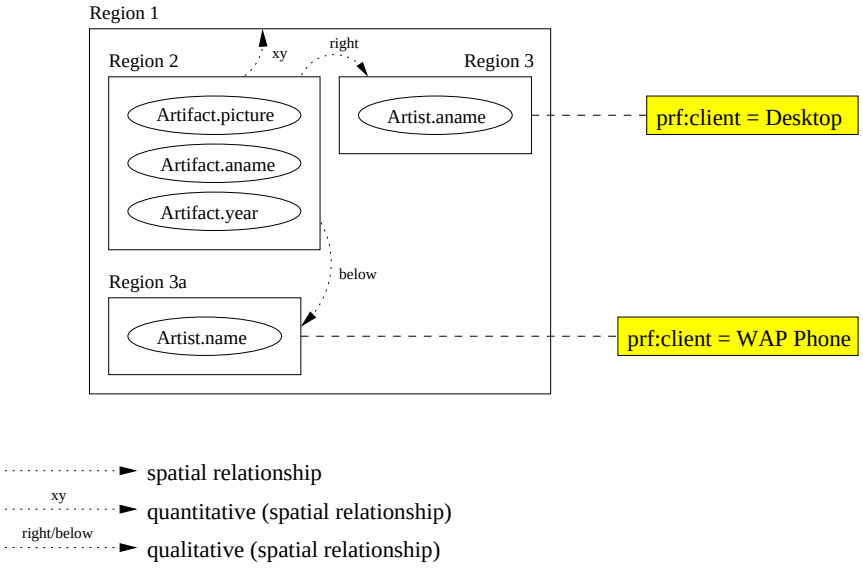
The OOHDM (*Object-Oriented Hypermedia Design Method*) approach proposed by Schwabe et al. [36] leverages object-oriented design primitives with a syntax close to that of UML for the design of Web application models. The OOHDM design process is articulated into a *concep-*



(a) Conceptual model.



(b) Application model.



(c) Presentation model.

Figure 2.8: Adaptation in the Hera method [42].

tual model, a *navigation model*, and a *interface model*. The conceptual model represents domain objects, relationships, and the intended application's functionality. The navigation model (expressed by a so-called *context schema*) defines nodes as views on conceptual objects (using a language similar to object database view definitions) that provide users with navigable objects. The (abstract) interface model describes the user-perceptible manifestation of navigation objects.

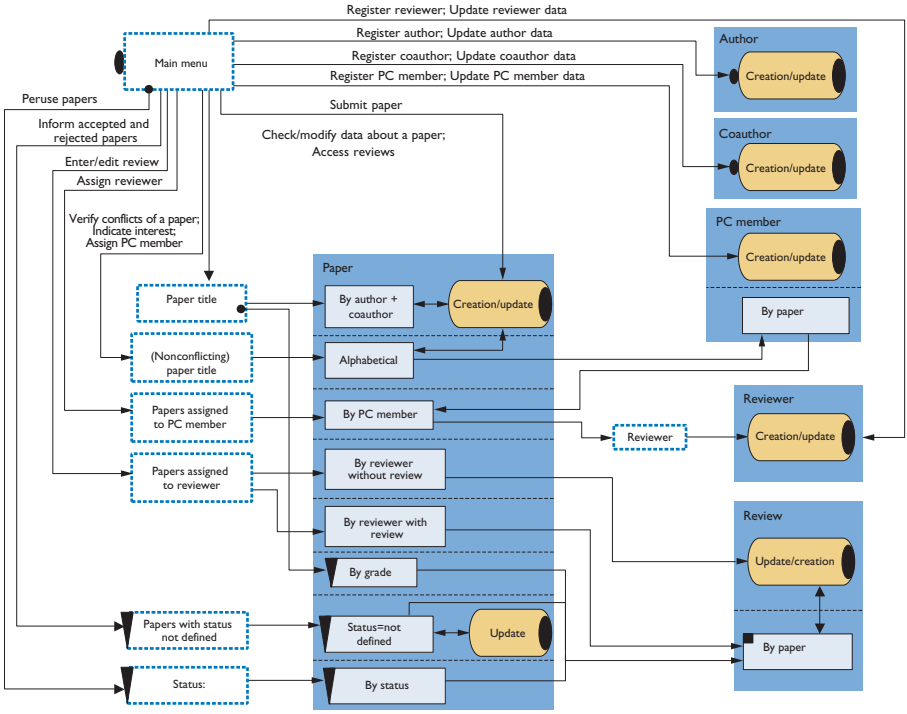
Customization or personalization in OOHDM is supported in the three design models in the following ways [43]:

- In the *conceptual model*, by explicitly representing users, roles, and groups, and by defining algorithms that implement different business rules for different users.
- In the *navigational model*, by defining different applications for each profile, by customizing node contents and structure, and by personalizing links and indexes.
- In the *interface model*, by defining different layouts according to user preferences or selected device characteristics.

In [43] the authors discuss personalization in OOHDM by means of an example conference paper review system (see Figure 2.9(a) for the described context diagram for navigating application concepts). More precisely, the authors discuss a set of recurrent personalization patterns and their specification in OOHDM: role-based personalization, link personalization, structure personalization, content personalization, and behavior personalization. Figure 2.9(b), for example, provides an example of role-based personalization for the application described in Figure 2.9(a). The navigation nodes presented to users are customized by querying the conceptual model according to a user's role: the public, customized view of papers augments the attributes from the `Paper` class with the paper's schedule, obtained by querying the `Session` object. The PC chair view, instead, adds the list of reviewers to each listed paper.

2.4.3 OO-H

OO-H (*Object-Oriented Hypermedia* [37]) is an object-oriented design model, which is based on two complementary application views, namely (i) the Navigational Access Diagram (NAD), which enriches the standard UML class diagram with navigation and interaction properties, and (ii) the Abstract Presentation Diagram (APD), which represents both the structure of the site and presentation details.



(a) OOHDM context diagram. Square solid boxes represent related objects, arrows indicate navigation possibilities, and dashed boxes are indexes.

```
NODE Public.Paper FROM Paper: p
        Name:String
        Authors: Set Set….
        Abstract: String
        Schedule
                Select date From Session:s
                Where p isScheduledIn s
```

```
NODE PC.ChairPaper FROM Paper: p
        Name:String
        Authors: Set Set….
        ….
        Reviewers:
                Set Select name From Reviewer:r
                Where p isReviewedBy r
```

(b) Role-based personalization in OOHDM. Different profile specifications yield different rendered information.

Figure 2.9: Personalization in the OOHDM method [43].

In the context of the OO-H methodology, Garrigós et al. [44] present a structured approach to personalize Web sites, based on three facets of a user profile, namely *user characteristics*, *user requirements*, and *user context* (see Figure 2.10(a)).

For each user profile, a personalization strategy is expressed by means of personalization rules addressing the application content and the navigation model for both single users and for groups of users. The rules can be expressed using the PRML language (Personalization Rule Modeling Language), complying with the ECA paradigm. Rules can be of three different types:

- *Acquisition rules* refer to where and what information must be acquired for enacting the personalization.
- *Personalization rules* specify the effect that the personalization causes on the system.
- *Profile rules* classify users according to the information they access.

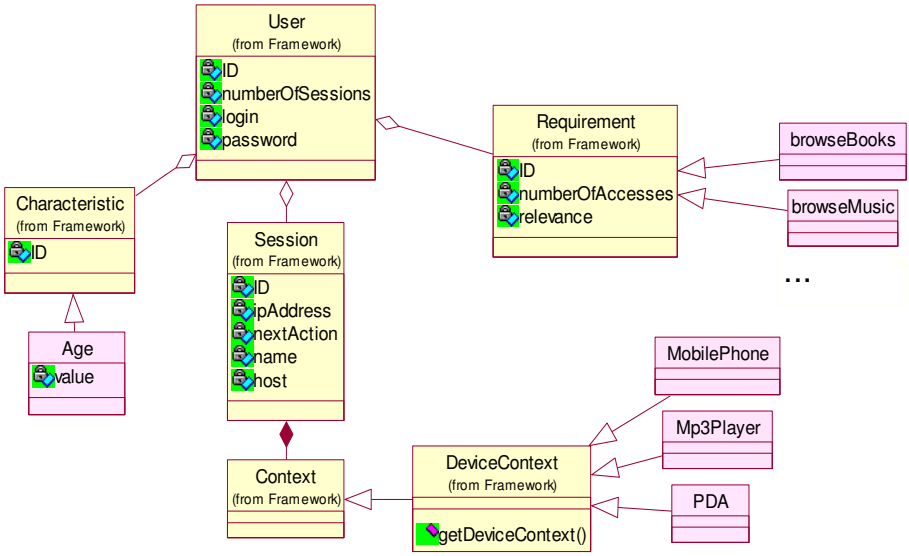
As reported in Figure 2.10(b), personalization is embedded in the Navigation Access Diagram. NADs are conceptual models that represent how users can navigate the Web application contents and services. In order to specify personalization, NADs also include the specification of events (*Start* events, *Navigation* events, *Method Invocation* events) that trigger the personalization rules, as well as the criteria that must be satisfied to fulfill the personalization requirements.

The whole set of rules associated to a NAD is stored in a PRML file. An example of such file is reported in Figure 2.11.

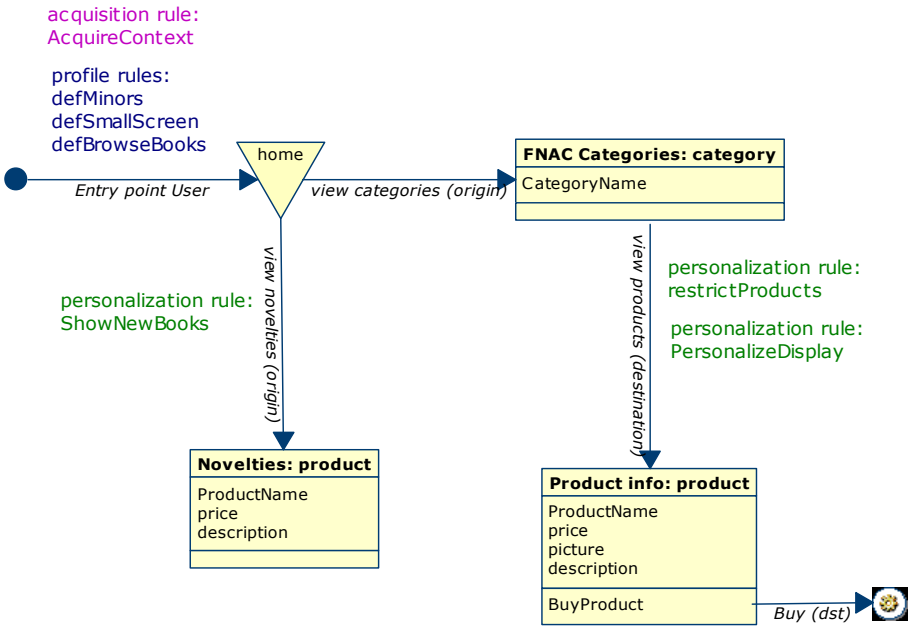
2.4.4 WSDM

WSDM (*Web Site Design Method*) is an *audience-driven* Web site design method: since the very first phases of design, the method puts the emphasis on the identification of the needs and characteristics of different classes of the target audience [38]. The addressed audience requirements refer to the information contents, functions, navigation structures, and usability. Such requirements, initially collected and specified, are then taken into account during the WSDM design activities, which address *Information Design*, *Functional Modeling*, and *Navigational Design*.

In [45] the authors present an extension of WSDM to cover the specification of adaptive behaviors. In particular, an event-based Adaptive Specification Language is defined, called ASL, which allows designers to express adaptations on the structure and the navigation of the



(a) OO-H user profile for personalization.

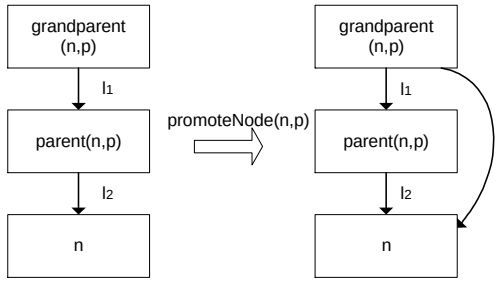


(b) OO-H Navigation Access Diagram.

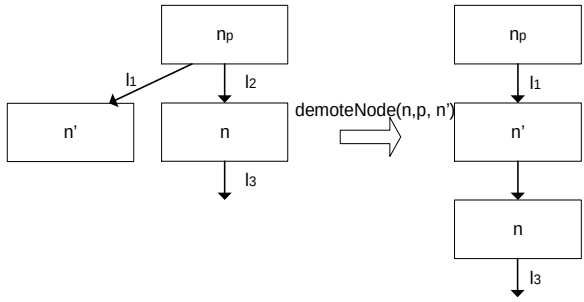
Figure 2.10: Personalization in the OO-H method [44].

```
# ACQUISITION SECTION
#RULE:"AcquireContext" priority="high"
When start do
DeviceContext=getDeviceContext()
endWhen

# PROFILE SECTION
#RULE:"defMinors"                    #RULE:"defSmallScreen"
priority:"medium"                    priority:"medium"
When start do                        When start do
   If (charHasValue (Age,<,18))       If (deviceContext="PDA" or
  then                               deviceContext="MP3" or
   AttachUserToPGroup("Minors")      deviceContext="WAP") then
 endIf                                AttachUserToPGroup("smallScreen")
endWhen                               endIf
                                     endWhen

#RULE:"defBrowseBooks" priority:"medium"
When start do
 If (userhasReq ("brosweBooks")) then
  AttachUserToPGroup("browseBooks")
 endIf
endWhen

# PERSONALIZATION SECTION
# RULE:"restrictProducts"            # RULE:"PersonalizeDisplay"
When Navigation.ViewProducts do      When Navigation.ViewProducts do
   If                                   Product.picture.Visible=false
   (Products.allowedFromAge>18)      endWhen
   then
        Product.Visible=false
   endIf
endWhen

# RULE:"ShowNewBooks"
When Navigation.ViewNovelties do
    If (Products.category='Books' and
    Products.dateOfAddition=currentDate-week) then
        Select(name, description, price) in Products
    endIf
endWhen
```

Figure 2.11: Example PRML rules in OO-H [44].



(a) Promotion of a node n along a path path (l1, l2).



(b) Demotion of a node n via a path p to a sibling n'.

Figure 2.12: Transforming the navigation structure in WSDM [45].

```
<rule> ::= ( <expression> ( ; <expression>) * |
           forEach ( <set> ( , <set> )* :
                     <expression>( ; <expression>)* )   )

<set> ::= <variable> ( , <variable>)* in <elements>
                     ( ! |with <property> ( and <property>) *  )

<elements> ::= ( Nodes | { node (, node)* } |
                 Chunks | { chunk (, chunk)* } |
                  Links  | { link (, link)* } )

<property> ::= <condition>

<expression>::= if <condition>
                 then <operation> ( ; <operation> )*

<operation> ::= ( deleteNode (node) | addNode (node) |
                 connect(chunk,node) | disconnect(chunk,node)|
                 deleteLink (link) | addLink (link) |
                 promoteNode(node, path) |
                 demoteNode(node1, path, node2) |
        linkNodes(node1, node2) |unLinkNodes(node1, node2) |
                 clusterChunks( node, chunk1, chunk2) |
                 clusterNodes( node, node1, node2)  )
```

Figure 2.13: BNF of the Adaptation Specification Language [45].

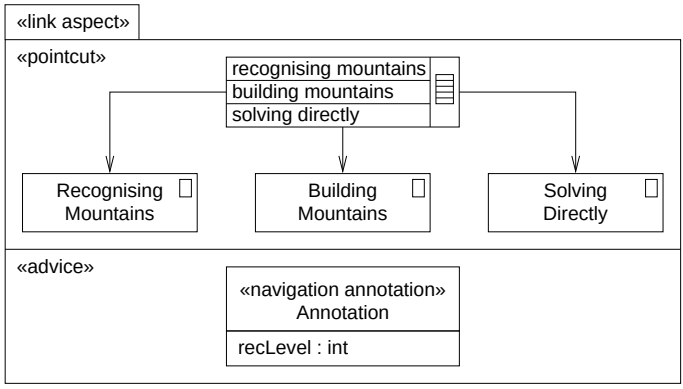
Web site. Such adaptations consist in transformations of the navigation model, which can be applied to nodes (*deleting/adding nodes*), information chunks (*connecting/disconnecting chunks* to/form a node), and links (*adding/deleting links*). Other transformations on the structure of the navigation model are described in Figure 2.12. In particular, *Promotion* (Figure 2.12(a)) makes a node easier to find by moving it closer to the root of the site; *Demotion* (Figure 2.12(b)) on the contrary moves the node farther away from the root. These adaptations may take place in function of the node popularity, following the rule "the more popular the node, the closer to the root".

It is worth noting that in WSDM the audience-driven paradigm also characterizes the adaptation features that are supported. Adaptation rules impact on the site structure depending on the usage of the Web site by the whole set of users and do not address individual user preferences.

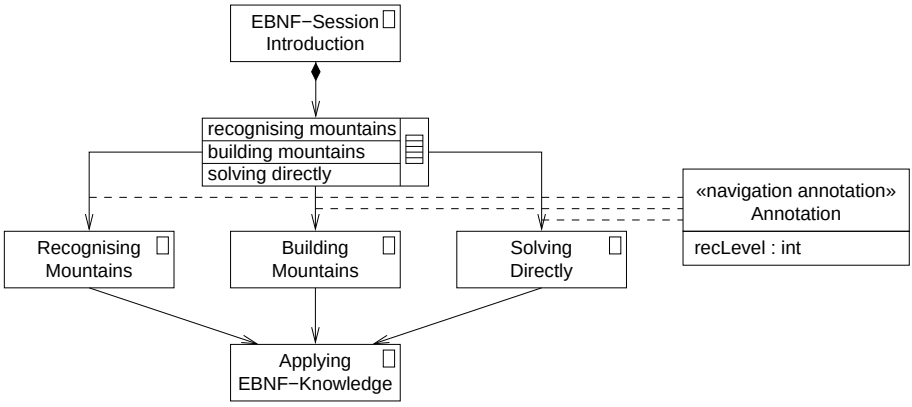
Figure 2.13 reports the BNF definition of the ASL language and some examples of rules expressed with this language.

2.4.5 UWE

UWE (*UML-based Web Engineering* [39]) is a UML-compliant design model defined as UML profile and as extension of the UML meta-model. It is characterized by the separate modeling of Web application concerns. Different models, based on UML stereotypes, are built for the *content*, the *navigation structure*, the *business processes*, and the *presentation*.



(a) Annotation model aspect.



(b) Weaving result.

Figure 2.14: Adding annotations to the UWE navigation structure [46].

Baumeister et al. [46] explore Aspect-Oriented Programming [47] techniques for modeling adaptivity in the context of the UWE method. The authors concentrate mainly on aspects for adaptive link hiding, adaptive link annotation, and adaptive link generation. This kind of adaptation is achieved in UWE by adding annotations to navigation links in the navigation model. Annotations are handled as independent model aspects; the authors distinguish model time and runtime aspects.

Figure 2.14 shows how annotations are introduced by using a model aspect that separates the annotation feature from the navigational behavior, thus documenting the navigation adaptation in a dedicated manner. In particular, as can be seen in Figure 2.14(a), the *pointcut* of the `«link aspect»` package describes the parts of the navigation model that are subject to adaptation. The *advice* then specifies that the class `Anno-`

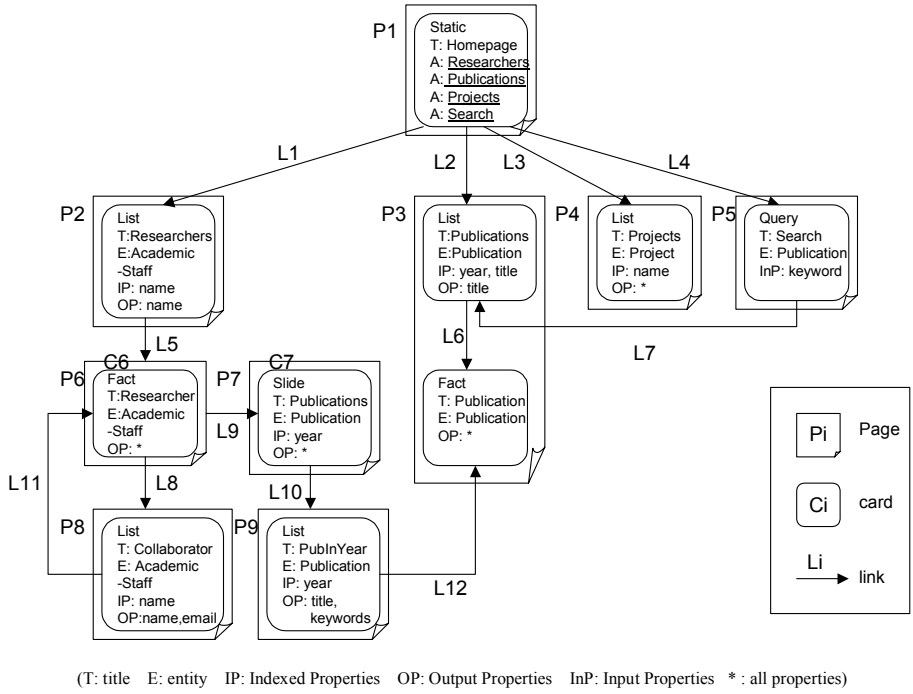


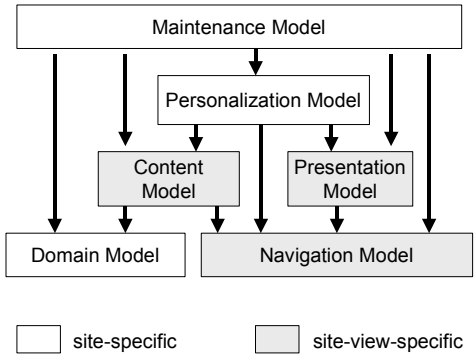
Figure 2.15: A site view graph in OntoWebber [40].

`tation` must be added to all the links included in the pointcut. The resulting model, merging navigation and adaptation, is reported in Figure 2.14(b).

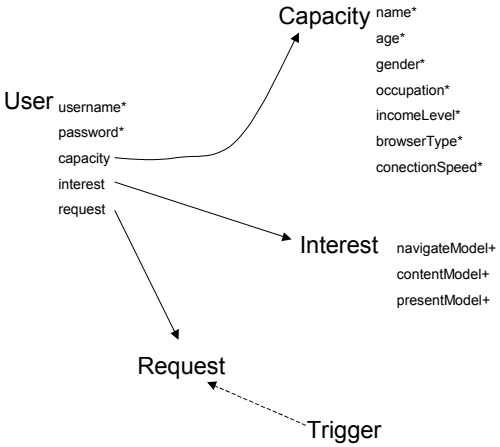
Content adaptation and presentation adaptation are not tackled yet in UWE, but they are under investigation by the authors. The main contribution of the work can be identified in the strong separation of navigation model and adaptation model, achieved by interpreting adaptation as cross-cutting aspect with respect to application modeling.

2.4.6 OntoWebber

Jin et al. [40] propose a fully ontology-based, model-driven approach to the declarative design of Web sites: OntoWebber. The authors stress the importance of data integration in the context of Web portals by means of semistructured data formats and equip OntoWebber with a suitable integration layer. Web site design with OntoWebber consists in the specification of a set of different models: *domain* model, *navigation* model, *content* model, *presentation* model, *personalization* model, and *maintenance* model, where the joint design of the navigation, content,



(a) Relationships between OntoWebber models.



(b) Personalization model.

Figure 2.16: Personalization in the OntoWebber method [40].

and presentation models is called *site view* design. In particular, site view design yields a so-called *site view graph*, which allows the automatic generation of the three different conceptual models. Figure 2.15, for example, describes a possible OntoWebber site view graph.

Figure 2.16(a) shows the relationships between the different design models in the OntoWebber approach. As showed in the figure, the content model, navigation model, and presentation model (i.e. the site view) can be customized according to the personalization model. The authors distinguish between *fine-grained* and *coarse-grained* adaptation/personalization; the former is achieved by incrementally rewriting a user's site view structure, the latter is achieved by switching between site views. Personalization is based on three user properties, i.e. *capacity*, *interest*,

$$\alpha_1((\varphi_0\alpha_2+\varphi_1\alpha_3(\alpha_5+1)\varphi_3+\varphi_2\alpha_4(\alpha_6+1)\varphi_4)^*\varphi_5)(\alpha_7\varphi_6+\alpha_{13}\varphi_7)$$
$$(\varphi_6\alpha_8(\alpha_8+1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8+\varphi_7\alpha_8\alpha_8^*\alpha_{14}\alpha_{15}\alpha_{16}^*\alpha_{11}\alpha_{17}(\overline{\varphi_{12}}\alpha_{18}\alpha_{19})^*\varphi_{12}\alpha_{18}\varphi_9)$$
$$\alpha_{20}(\varphi_{10}+\varphi_{11})$$

(a) SiteLang story space expressed by means of a *many-sorted Kleene algebra with tests*.

$$\varphi_5 x = x \text{ for all } x \in K$$
$$\varphi_5\varphi_0 = 0 \qquad \varphi_5\varphi_1 = 0 \qquad \varphi_5\varphi_2 = 0$$

(b) Personalization predicates.

$$\alpha_1(\alpha_7\varphi_6+\alpha_{13}\varphi_7)$$
$$(\varphi_6\alpha_8(\alpha_8+1)\alpha_9\alpha_{10}\alpha_{11}\alpha_{12}\varphi_8+\varphi_7\alpha_8\alpha_8^*\alpha_{14}\alpha_{15}\alpha_{16}^*\alpha_{11}\alpha_{17}(\overline{\varphi_{12}}\alpha_{18}\alpha_{19})^*\varphi_{12}\alpha_{18}\varphi_9)$$
$$\alpha_{20}(\varphi_{10}+\varphi_{11})$$

(c) Simplified/personalized application model.

Figure 2.17: Personalization in the SiteLang method [48].

and *request*; Figure 2.16(b) shows the adopted personalization model.

2.4.7 SiteLang

SiteLang [41] is a storyboard specification language that allows the specification of information services based on the concepts of story and interaction spaces as well as media objects. More precisely, SiteLang is a process algebra for storyboarding, i.e. for expressing application "stories".

In the context of SiteLang, Schewe et al. [48] describe a singular, algebraic approach to personalization or adaptation in Web information systems by leveraging Kleene algebras with tests [49] as alternative formalization of SiteLang story spaces. Figure 2.17(a), for example, shows an algebraic expression representing a story space. The so formalized story space can now be personalized according to the preferences of application users by specifying proper (conditional) equations, as exemplified in Figure 2.17(b). User preferences are thus specified by means of proper pre- or post-conditions that act as filters over the Web information system's story space and that tailor the algebraic expression of the story space to the individual user. Applying the algebraic preferences to the expression of the story space enables the automatic simplification of the story space into the expression shown in Figure 2.17(c). Unlike the previous conceptual approaches, this idea heavily leverages formal reasoning about Web information systems.

In [50] the authors show how SiteLang applications in general can be adapted to the usage history, context parameters (e.g. the current delivery channel), and/or to the role users play in the application.

2.4.8 Comparison of Approaches

As the previous description of design methods shows, the different focuses of the most prominent conceptual Web design methods and their different modeling paradigms yield different levels of support for application adaptation. Adaptation features are expressed by means of a variety of formalisms, and, also, the actual adaptation features supported vary from method to method. For instance, while all of the described methods support some form of personalization, only few of them are designed to support more advanced adaptation mechanisms, i.e. runtime adaptivity or context-awareness.

To compare the described design methods more systematically, we introduce a set of dimensions that we believe are apt to express how adaptation is supported by those methods. In particular, the dimensions are:

- *Adaptability*: does the approach provide means for the design time adaptation to device characteristics or user profiles?
- *User profile adaptivity*: does the approach provide means for the runtime adaptation to a user model?
- *Context-Awareness*: does the approach support adaptivity according to a generic and dynamic context model?
- *Modeling Paradigm*: how is application adaptation (i.e. rules) specified?
- *Tool support*: is the approach supported by a proper CASE tool, assisting designers throughout the development phases?

Table 2.1 summarizes how the discussed design methods address the identified dimensions. In general, all of the approaches support the (static or design time) *adaptability* of the application to individual user profiles and/or device characteristics. *Adaptivity* and *context-awareness* are supported only in a very limited fashion, especially context-awareness is not addressed as envisioned in the introductory sections of this chapter.

WSDM does not explicitly support context-awareness, but the proposed approach to application adaptation, which distinguishes WSDM from the approaches of the other methods, leads us to say that WSDM

presents context-aware features: its audience-driven adaptation of the application is in fact not triggered by an individual user model (as required by the above characterization), but by a model of the overall audience of the application, i.e. a context property according to our definitions. In addition, the effects of adaptation actions (e.g. the elimination of a navigation node) effectively apply to the application as a whole in a global fashion, as opposed to user-specific (i.e. individual) adaptation effects supported by the other methods.

The PRML language proposed by Garrigós et al. in the context of the OO-H method provides a singular, unified solution by means of runtime personalization rules addressing both adaptability and adaptivity; typical design time decisions, such as those concerning adaptability, are thus handled at runtime. Although this aspect could have the disadvantage of delaying adaptability decisions to the execution time, the integration of OO-H and PRML seems to offer the most complete adaptation coverage.

Concerning the *modeling paradigms* underlying the methods, they all count on more or less intuitive graphical models in the actual application design process. Adaptation design is however mostly addressed in a textual fashion, requiring the designer to get familiar with a different specification paradigm. In a unique fashion, SiteLang is characterized by a very strong formalization, which allows the application to leverage automatic reasoning techniques to personalize the system's story space; a similar formal paradigm (based on *personalization predicates*) is kept also for personalization specification.

Tool support for the computer-assisted development of Web applications is not provided by all of the described methods. Especially adaptation specification, which may require the designer to hand-code adaptation rules and to switch from the original to another modeling paradigm, is out of the scope of the tools.

Due to the lack of an explicit support for context-awareness, none of the approaches discussed leverages a proper context model for application adaptation. However, in principle all methods could be extended to support adaptivity and context-awareness. In general, we observe that the more an approach also provides an implementation framework (not only a design instrument), possibly also shaping the architecture of the final application, the more application adaptation can be shifted from pure design time to runtime, thus enabling dynamic adaptation mechanisms.

Throughout this dissertation, we will introduce concepts and solutions that address the five dimensions used to characterize adaptivity. More precisely, we will show how the WebML modeling language, properly

extended, is able to cope with *adaptability*, *adaptivity*, and *context-awareness*. We will try to keep as much as possible the *modeling paradigm* that already characterizes WebML and to extend the WebML *CASE tool* accordingly.

Table 2.1: Comparison of the adaptation features of the most prominent conceptual design methods.

Method	Dimension	Evaluation
Hera	Adaptability	It fully supports adaptability with respect to device characteristics and user preferences by means of *appearance conditions* attached to modeling constructs.
	Adaptivity	There is no explicit notion of adaptivity; it is however considered future work [42].
	Context-Awareness	Adaptability features are based on CC/PP profiles, and there is no support for a more general context model. If adaptivity is introduced in Hera, it should however be easy to leverage also general context data.
	Modeling Paradigm	Hera models adopt an intuitive, graphical modeling paradigm; adaptability rules are specified through a textual formalism.
	Tool Support	The method is equipped with a proper Hera Presentation Generator [51], based on Hera design models.
OOHDM	Adaptability	User preferences and device characteristics can be taken into account by tailoring *context diagrams*.
	Adaptivity	There is no explicit support for runtime application adaptation.
	Context-Awareness	Customization is considered with respect to user profile and device, more general context data is not yet discussed.
	Modeling Paradigm	OOHDM models are inspired by the object-oriented design paradigm (e.g. UML). Customization of nodes, links and indexes is expressed by queries over conceptual objects.

continued on next page

continued from previous page

Method	Dimension	Evaluation
	Tool Support	A template-based environment, OOHDM-Web [52], supports site development.
OO-H	Adaptability	Adaptability can be specified by means of the PRML rule language.
	Adaptivity	Adaptivity is supported through adaptation rules and a proper runtime profiling mechanism.
	Context-Awareness	OO-H supports context-based personalization through explicit context representation, addressing location, network status, device type, and time.
	Modeling Paradigm	The application design is expressed by means of UML-based visual diagrams. Adaptation rules are expressed by means of textual PRML rules bound to visual diagrams by specifying the triggering events.
	Tool Support	OO-H development is supported by the VisualWade CAWE tool [53].
WSDM	Adaptability	WSDM does not explicitly provide adaptability to user preferences and devices.
	Adaptivity	Not explicitly supported.
	Context-Awareness	Application adaptations are triggered by a model of the whole community of users and thus "transform" the navigation model based on the global Web site usage.
	Modeling Paradigm	WSDM models are visual, while adaptivity is expressed by means of textual ASL rules.
	Tool Support	The Audience Modeler CASE tool supports the audience modeling phase of WSDM, the Chunk Modeler CASE tool supports the data modeling phase.
UWE	Adaptability	Adaptability of contents, links, and presentation properties is supported by means of a representation of user characteristics (e.g. task, preference, interests).
	Adaptivity	Adaptive link hiding and adaptive link generation are managed at runtime based on the current status of the user model.
	Context-Awareness	UWE covers context representation, addressing user location and hardware, software, and network platform.

continued on next page

continued from previous page		
Method	Dimension	Evaluation
	Modeling Paradigm	The application design uses UML-based visual diagrams. Adaptation is based on aspect-oriented modeling by separating cross-cutting features into *aspects* bound to the principal model.
	Tool Support	The UWE method is supported by ArgoUWE, which does not include support for adaptivity. Some tool extensions are however planned.
OntoWebber	Adaptability	The method supports coarse-grained adaptation by tailoring site views to user groups and fine-grained adaptation based on a predefined personalization model.
	Adaptivity	Adaptivity at runtime is not mentioned.
	Context-Awareness	Design support for context-aware application features is not provided.
	Modeling Paradigm	Ontologies (modeled in RDF and DAML+OIL) are the grounding of OntoWebber. The site view graph is an intuitive, graphical schema, that can be adapted to the needs of the users.
	Tool Support	OntoWebber is equipped with a proper development suite [54], comprising Ontology Builder, Site Builder, Site Generator, and Personalization Manager.
SiteLang	Adaptability	Support for adaptability to preferences and devices characteristics is provided.
	Adaptivity	In [50] the authors use the term *adaptivity*, but sometimes with the meaning of *adaptability*, according to our definition.
	Context-Awareness	In [50] the authors show how context is used to suitably select media types.
	Modeling Paradigm	The main idea if SiteLang is story boarding, based on the modeling of usage scenarios. SiteLang has a strong formalization, also enabling automatic reasoning.
	Tool Support	SiteLang development is aided by the Storyboard Editor [55].

2.5 Discussion

This chapter provided some insight into the historical perspective of context-awareness and of context modeling in general. We discussed some works that are related with the approach we will develop through-

out this dissertation, i.e. we discussed the most prominent conceptual Web design methods. Hence, special focus was put on design support for context-awareness or adaptivity in Web applications. The comparison of the methods shows that although all of the methods provide means for the design of adaptable Web sites or for personalization, none of them explicitly supports the design of context-aware Web applications, where adaptivity may be triggered by a generic context model during the execution of the application.

In order to support the development of context-aware Web applications by means of conceptual modeling methods, we therefore need to enrich the expressive power of the adopted models and/or languages, so as support the modeling of context data and the definition of adaptive application features. Showing how this can be done in WebML, is the aim of this dissertation.

3 The Web Modeling Language (WebML)

The Web Modelling Language (WebML) is a third-generation Web design methodology conceived in 1998 on the wake of the early hypermedia models and of the pioneering works on Hypermedia and Web design, like HDM [56] and RMM [57].

The original goal of WebML was to support the design and implementation of so-called data-intensive Web applications [1], defined as Web sites for accessing and maintaining large amounts of structured data, typically stored as records in a database management system, like on-line trading and e-commerce applications, institutional Web sites of private and public organizations, digital libraries, corporate portals, and community sites. But the language has also constantly been subject to evolution, and the original version has been extended with new concepts and modeling facilities to cope with novel requirements that have been posed to Web applications. For instance, WebML has been extended toward the modeling of Web services and the interaction between a Web application and Web services [58] and toward the modeling of process-enabled Web applications [59].

The extension of WebML toward the modeling of context-aware and adaptive Web applications described in this dissertation requires knowledge of WebML-specific concepts and notions. Therefore, this chapter introduces the reader to the necessary terminology. Note that the contents of this chapter are outside the scope of the research that led to this dissertation.

3.1 Introduction

WebML is a visual language for the specification of the structure of a Web application and the organization and presentation of contents in one or more hypertexts. For the definition of its visual modeling primitives, WebML reused existing conceptual data models and proposed an original notation for expressing the navigation and composition features

of hypertext interfaces. The hypertext model of WebML took an approach quite different from previous proposals: instead of offering a high number of primitives for representing all the possible ways to organize a hypertext interface that may occur in data-intensive Web applications, the focus was on inventing a minimal number of concepts, which could be composed in well-defined ways to obtain an arbitrary number of application configurations. This initial design choice deeply influenced the definition of the language and its evolution toward more complex classes of applications.

Four major versions of WebML characterize the progression of the language:

- *WebML 1*: The original version comprised only a fixed set of primitives to model read-only data-intensive Web sites. The focus was on the modular organization of the interface, on navigation definition and on content extraction, and publication in the interface.

- *WebML 2*: It added support to model business actions (called operations), triggered by the navigation of the user. In this way, the expressive power was extended to support features like content management and authentication and authorization.

- *WebML 3*: The introduction of the concept of model plug-ins transformed WebML into an open language, extensible by designers with their own conceptual-level primitives, so to widen the expressive power to cover the requirements of new application domains. This transition emphasized the role of component-based modeling and was the base of all subsequent extensions.

- *WebML 4*: The notion of model plug-ins was exploited to add orthogonal extensions to the core of WebML, covering sectors and applications not previously associated with model-driven development. For example, Web Service interaction and workflow modeling primitives were added as plug-in components to enable the modeling and implementation of distributed applications for multi-actor workflow enactment [58, 59].

A distinctive trait of the WebML experience is the presence of an industrial line of development running in parallel to the academic research. One of the original design principles of WebML was implementability, with the ultimate goal of bringing Model-Driven Development (MDD) to the community of "real" developers. To achieve this objective, Politecnico di Milano spun off a company (called Web Models) in 2001, with

the mission of implementing and commercializing methods and tools for model-driven development of Web applications, based on WebML.

The major result of the industrial R&D is WebRatio [2], an integrated development environment supporting the modeling of applications with WebML and their implementation with model-driven code generators. Today WebRatio is a consolidated industrial reality: more than one hundred applications have been developed by Web Models' customers, over 4.000 trial copies are downloaded per year, and many universities and institutions in the world use the tool in their Web Engineering courses.

3.2 WebML Design Overview

The WebML design process starts with the definition of a data schema, expressing the organization of the application content. The WebML *WebML!Data Model* adopts the Entity-Relationship (ER) primitives to represent 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 designers to describe 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 in 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, to serve 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 or similar appliances. 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 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 (except 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 to determine 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 composite navigation mechanisms. Links between units are used to carry pieces of information from the source unit to the destination unit.

- The *operation model*, consisting of a set of units for specifying content management *operations*. The basic primitives support 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).

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.

3.3 Data Model

The goal of data modeling is to enable the specification of the data used by the application in a formal yet intuitive way. The result of data modeling is a conceptual schema, which conveys in a simple and readable way the available knowledge about the application data. WebML does not propose yet another data modeling language, but exploits the most successful and popular notation, namely the Entity-Relationship (ER) model.

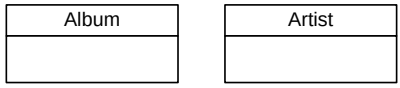


Figure 3.1: Graphic notation for entities.

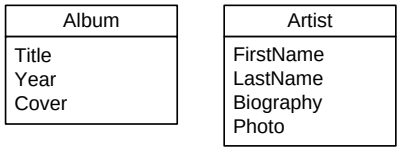


Figure 3.2: Graphic notation for entities and attributes.

3.3.1 Entities

Entities are the central concept of the Entity-Relationship model. An entity represents a description of the common features of a set of objects of the real world. Examples of entities are `Person`, `Car`, `Artist`, and `Album`. An entity has a population, which is the set of objects that are described by the entity. These objects are also called the instances of the entity.

As all the concepts of the Entity-Relationship model, entities are specified using a graphic notation. They are denoted by means of rectangles, with the entity name at the top. Figure 3.1 shows an Entity-Relationship schema consisting of two entities: `Album` and `Artist`.

3.3.2 Attributes

Attributes represent the properties of real world objects that are relevant for the application. Examples of attributes are the name, address, and photo of a person. Attributes are associated with entities, with the meaning that all the instances of the entity are characterized by the same set of attributes. In other words, the entity is a descriptor of the common properties of a set of objects, and such properties are expressed as attributes.

Attributes are graphically represented inside the entity box, below the entity name, as shown in Figure 3.2. In the example, the entity `Album` is characterized by attributes `Title`, `Year` and `Cover`, and the entity `Artist` by attributes `FirstName`, `LastName`, `Biography` and `Photo`.

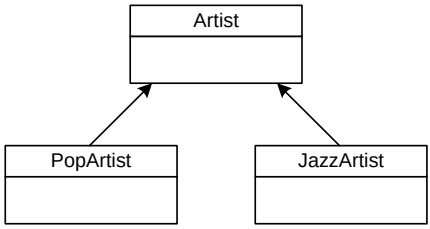


Figure 3.3: Generalization hierarchy.

3.3.3 Identification and Primary Key

All the instances of an entity must be distinguishable by means of a unique identity that permits their unambiguous identification. It is good practice to define the primary key of entities using a single special purpose attribute, called object identifier (abbreviated as OID), whose sole purpose is to assign a distinct identifier to each instance of an entity. WebML assumes that the OID property is implicitly defined for all entities and omits it from the Entity-Relationship diagrams.

Alternative identification schemas are admissible and the identifying attributes can be defined as keys (also called alternative keys). Alternative keys must be not null and unique, just like primary keys.

3.3.4 Generalization Hierarchies

The Entity-Relationship model permits the designer to organize entities into a hierarchy, where they share common features. The basic generalization hierarchy (also called IS-A hierarchy) has one super-entity and one or more sub-entities (see Figure 3.3 for an example). Each sub-entity inherits all attributes and relationships defined in the super-entity and may add locally-defined attributes and relationships.

3.3.5 Relationships

Relationships represent semantic connections between entities, like the association between an artist and his/her album or between an artist and his/her reviews. The meaning of the association is conveyed by the relationship's name, which is established by the designer. The simplest form of relationship is the binary relationship, which connects two entities, but also relationships involving more than two entities, called N-ary relationships, are allowed.

Relationships can be annotated with minimum and maximum cardinality constraints, respectively denoting the minimum and maximum

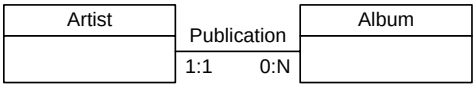


Figure 3.4: Graphic notation for relationships.

number of objects of the destination entity to which any object of the source entity can be related. Based on their maximum cardinality constraints, relationships are called "one-to-one", if both relationships roles have maximum cardinality 1, "one-to-many", if one relationship role has maximum cardinality 1 and the other role has maximum cardinality N, or "many-to-many", if both relationships roles have maximum cardinality N. Figure 3.4 shows a "one-to-many" relationship, associating one artist to multiple albums.

The Entity-Relationship model admits N-ary relationships and relationships with attributes. However, as well known in the data modeling field, both these constructs can be represented using a combination of entities and binary relationships, thus we do not investigate them more accurately.

3.4 Hypertext Model

The goal of hypertext modeling is to specify the organization of the front-end interfaces of a Web application. The specification of the hypertext in WebML is kept as much as possible at the conceptual level, which means that it does not commit to specific design or implementation choices. In fact, while drawing a WebML hypertext model, a developer does not already need to worry about, for example, the actual distribution of functionality between the various tiers of the Web application. These and other design choices may be taken later on in the development process.

3.4.1 Pages

Pages are the actual interface elements delivered to the user who browses the hypertext by accessing its pages in the desired sequence. A page typically consists of several units, grouped together to accomplish a well-defined communication purpose.

Figure 3.5 shows the graphic notation for pages, which is simply a labeled box surrounding the units that belong to the page. In the example, the page called `AlbumPage` contains two content units, one to display the list of all artists and one to display the list of all albums. The figure also

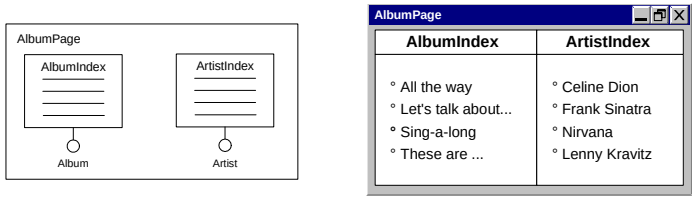


Figure 3.5: WebML graphic notation for pages.

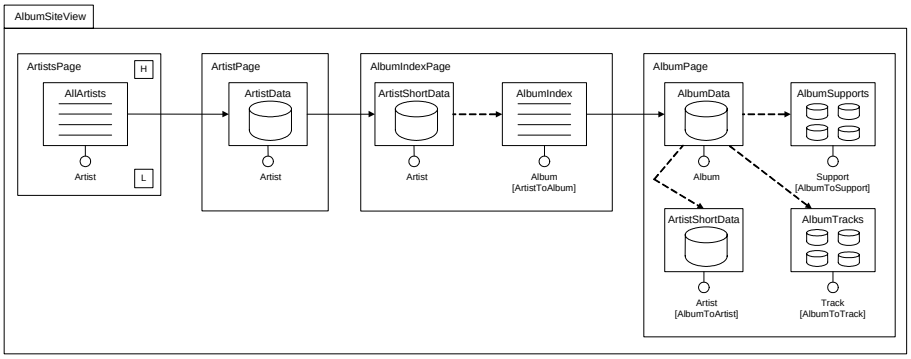


Figure 3.6: Site view in WebML.

shows a possible rendition of the `AlbumPage` in HTML, which is simply the aggregation of the renditions of the two units.

Note that, although in the figure the position of the units in the HTML rendition is the same as that of the index unit icons in the WebML specification, this fact is purely coincidental. A WebML page specification is abstract and has nothing to do with presentation aspects, like the relative position of content elements in the HTML rendition.

3.4.2 Hypertext organization

The specification of large and complex hypertexts, starting form pages, can be organized hierarchically, by using modularization constructs such as site views, areas, and nested pages.

Site Views

A WebML hypertext is packaged into an application to be delivered to users by enclosing its linked pages and units into a modularization construct called site view.

Figure 3.6 shows an example of the graphic notation for site views. Site views are characterized by a user-defined name and contain a set

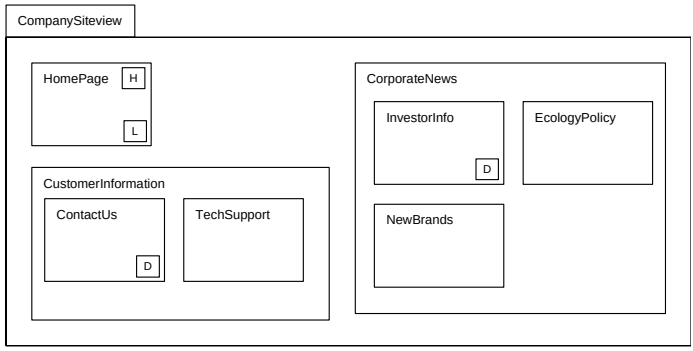


Figure 3.7: Two-level decomposition of site views into areas.

of pages and/or areas. In the example, the `AlbumSiteView` groups four pages into one module.

Areas, Landmarks and Home Pages

Like site views, areas as well are containers of pages or, recursively, other sub-areas, which can be used to give the site view a hierarchical organization. Links can be drawn between pages and units in the usual way, and can cross the borders of areas. Following an inter-area link simply implies that the focus moves from a page of one area to a page of another area.

Pages and areas are characterized by some distinguishing properties, which highlight their importance in the Web site. In particular, pages inside an area or site view may have the following three properties, which can be found graphically in Figure 3.7:

- The *home page* (small "H" inside the page icon) is the page at the (unique) default address of the site and presented after the user logs in to the application.
- The *default page* ("D") is the one presented by default when its enclosing area is accessed. The default page inside an area must be unique.
- A *landmark page* ("L") is reachable from all the other pages or areas within its enclosing module (the site view or a super-area) by means of a specific link pointing to that page.

Areas are associated with landmark and default properties, using the same notations as pages (see Figure 3.7):

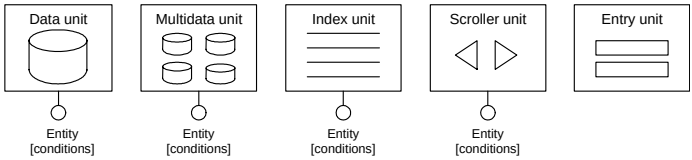


Figure 3.8: Graphic representation of the five basic units with source and selector conditions.

- Analogous to the previous definition, the default area is the subarea accessed by default when its enclosing super-area is accessed.
- A landmark area is an area implicitly reachable from all other pages or areas of the enclosing site view or a super-area.

Areas are not meant only for structuring several pages of a site view into conceptual hierarchies, but from an operational point of view, by means of such hierarchies it is possible to derive a menu structure valid for the overall site view, which permits easy access to landmark areas or pages.

3.4.3 Units

Units are the atomic elements for specifying the content of a Web page. WebML supports five types of units (cf. Figure 3.8):

- *Data units*, which show information about single objects.
- *Multidata units*, which present information about a set of objects.
- *Index units*, which show a list of descriptive properties of some objects, without presenting their detailed information.
- *Scroller units*, which enable the browsing of an ordered set of objects, by providing commands to access the first, last, previous, and next element of a sequence.
- *Entry units*, which model entry forms, whose fields allow the gathering of input, needed to perform searches or to feed update operations.

The five basic types of content units can be combined to represent Web pages of arbitrary complexity. The first four units model the publication of information to users, while entry units express the acquisition of information from users. Among the four units for information publishing,

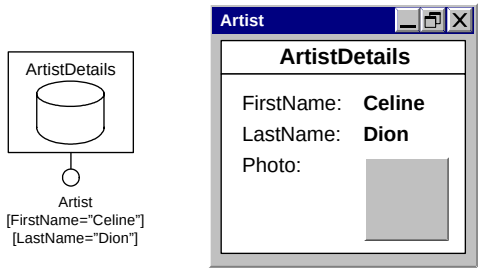


Figure 3.9: WebML graphic notation for data units and rendition in HTML.

data and multidata units present the actual content of the objects they refer to, whereas indexes and scroller units facilitate the selection of objects. Data units refer to a single object, whereas multidata, index, and scroller units refer to a set of objects.

Data, multidata, index, and scroller units present content extracted from the data schema; therefore, it is necessary to specify where their content comes from. WebML uses two concepts to express the origin of a unit's content, the source and the selector:

- The *source* is the name of the entity from which the unit's content is extracted. Thus, the source entity tells the type of the objects used to compute the unit's content.
- The *selector* is a predicate, used to determine the actual objects of the source entity that contribute to the unit's content. Selectors are the conjunction of elementary conditions, built from the entity attributes, from the relationship roles in which the entity is involved, and from constant or variable terms. Variable terms are constructed using parameters associated with the input links of the unit. Selectors whose conditions use parameters are called parametric selectors.

Data Units

Data units publish a single object of a given entity. A data unit is characterized by a user-defined name, a source, an optional selector, and the set of included attributes.

Figure 3.9 shows a data unit called `ArtistDetails`, with its source and selector highlighted below the icon. The unit is defined over the entity `Artist`, and shows the specific object determined by evaluating

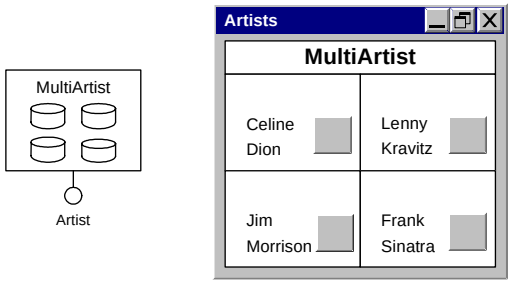


Figure 3.10: WebML graphic notation for multidata units, and rendition in HTML.

its selector, which is the conjunction of two equality-based predicates on attributes `FirstName` and `LastName`. The visualized attributes are not specified in the graphic notation, but are declared as unit properties.

The selector of the data unit includes a conjunction of two simple predicates. Besides conjunction, two forms of disjunction can be specified in a selector predicate:

- *Value disjunction*: a single attribute value is compared with a set of values using the expression [attribute operator value1 | value2 | ... | valueN]. This corresponds to the predicate ((attribute operator value1) OR (attribute operator value2) OR ... OR (attribute operator valueN)).
- *Attribute disjunction*: a set of attributes is compared with a single value using the expression [attribute1 | attribute2 | ... | attributeN operator value]. This notation corresponds to the predicate ((attribute1 operator value) OR (attribute2 operator value) OR ... OR (attributeN operator value)).

Multidata Units

Multidata units present multiple objects of an entity together by repeating the presentation of several data units. A multidata unit is characterized by a user-defined name, a source, an optional selector, the included attributes, and an optional order clause. See Figure 3.10 for the graphic notation for multidata units.

Index Units

Index units present multiple objects of an entity as a list and are typically used to select one particular object. An index unit specification includes

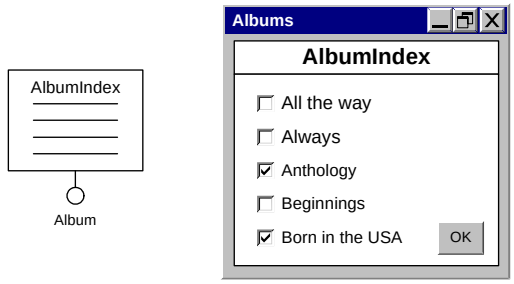


Figure 3.11: WebML graphic notation for index units and rendition in HTML.

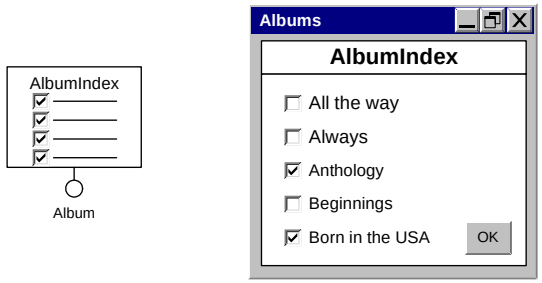


Figure 3.12: WebML graphic notation for multi-choice indexes, and rendition in HTML.

the user-defined name, the source, the optional selector, the included attributes, and the optional order clause. Figure 3.11 shows the graphic notation for index units.

Multi-choice Index Unit

A first variant of index unit is represented by the multi-choice index unit, in which each element of the list of entries is associated with a checkbox, allowing the user to select multiple objects, instead of a single one. The graphic notation for representing a multi-choice index unit and a possible rendition are depicted in Figure 3.12.

Hierarchical Index Unit

A second variant of index unit is the concept of hierarchical index, in which the index entries are organized in a multi-level tree. The hierarchy is represented by a sequence of N source entities connected by N-1 relationship roles. The first source entity represents the instances at the top level of the hierarchy; the second source entity, introduced in the selec-

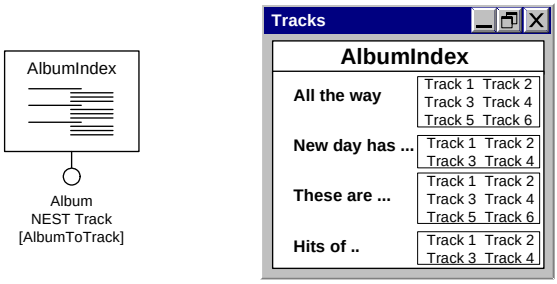


Figure 3.13: WebML graphic notation for hierarchical indexes and rendition in HTML.

tor by the NEST clause, represents the instances at the second level of the hierarchy, and so on. Each relationship role denotes the father-child association between two entities at consecutive levels in the hierarchy.

Selector conditions can be specified for the source entities at any level of the hierarchy, and even recursive relationships defined over certain entities are possible. See Figure 3.13 for an example graphic rendition of a hierarchical index unit.

Scroller Units

Scroller units provide commands to scroll through the objects in a set, for example to scroll over all the instances of an entity. A scroller unit specification is characterized by the user-defined name, the source, the optional selector, the block factor (i.e. the number of objects that are scrolled together), and the optional order clause.

Figure 3.14 shows the WebML graphic notation for representing a scroller unit and a possible rendition in an HTML-based implementation. The `AlbumScroll` unit is defined over the entity `Album` and has no selector; thus, it can be used for moving along the set of all albums. In particular, it is possible to move to the first, previous, next, and last album, according to the sorting clause specified in the unit.

Entry Units

Entry units support form-based data entry, as exemplified by Figure 3.15. They are used to gather input, which is typically employed to:

- Perform *searches* over the objects of an entity, for example to locate the instances of an entity whose attributes contain a given keyword.

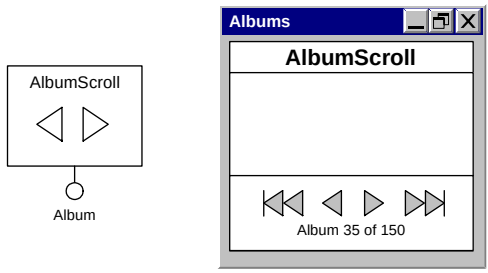


Figure 3.14: WebML graphic notation for scroller units, and rendition in HTML.

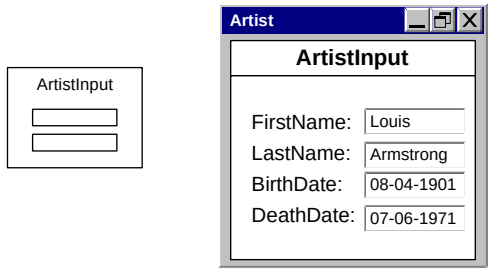


Figure 3.15: WebML entry unit, and rendition in HTML.

- Supply *parameters* to operations like content updates, login, and external services.

3.4.4 Links

Neither pages nor units exist in isolation, because real-world hypertexts are made of connected pages that contain several interrelated pieces of content and commands permitting the user to interact with the application. To express these features, pages and units can be linked to specify the allowed navigation paths between pages, the selections offered to the user and the effect of the user's interaction on the content of the units displayed in the page.

Navigation modeling is the part of hypertext modeling that deals with the specification of the links between units and pages, and of the properties of such links. The central notions of navigation modeling are the concepts of link, link parameters, and parametric selectors:

- A *link* is an oriented connection between two units or pages.
- A *link parameter* is the specification of a piece of information, which is transported from the source to the destination of the link.

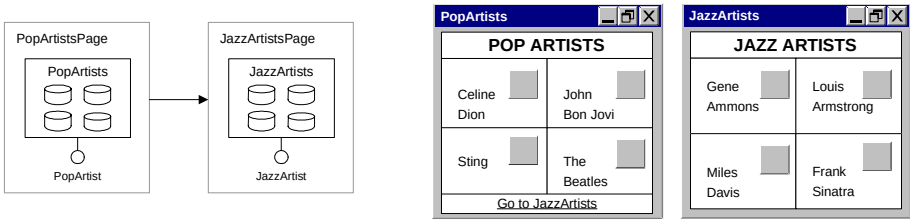


Figure 3.16: Non-contextual link connecting two pages.

- A *parametric selector* is a unit selector whose predicates contain a reference to a link parameter.

Specification of Links

Links abstract and generalize the fundamental notion of hypertexts: the concept of anchor. An *anchor* is an active device, through which the user can interact with the hypertext and which is classifiable as follows:

- Anchor tags, with an `href` attribute that refers to another page.
- Anchor tags, with an `href` attribute that refers to the same page.
- Confirmation buttons of HTML forms used for searching.
- Confirmation buttons of HTML forms used for sending input to operations, for example for logging into a password-protected site.

Essentially, links (i) enable the navigation of the hypertext by letting the user move from a source page to a destination page and (ii) transport information from one unit to another. In the WebML terminology, links crossing the boundaries of pages are called *inter-page* links, whereas links with the source and destination inside the same page are called *intra-page*; links transporting information are called *contextual*, in contrast with *non-contextual* links, which do not transport information. Graphically links are represented by oriented arcs, which connect the source unit or page to the destination unit or page.

The example of Figure 3.16 shows an inter-page, non-contextual link. The link connects a source page (`PopArtistsPage`), which includes a multidata unit showing pop artists, to a destination page (`JazzArtistsPage`), which includes a multidata unit showing jazz artists. The content of page `JazzArtistsPage` is independent from the content of page `PopArtistsPage`, and thus the navigation of the link does not require any information to be passed from the source to the destination page.

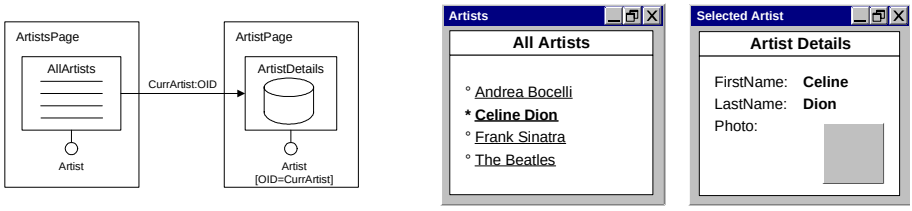


Figure 3.17: Inter-page contextual link with associated link parameter.

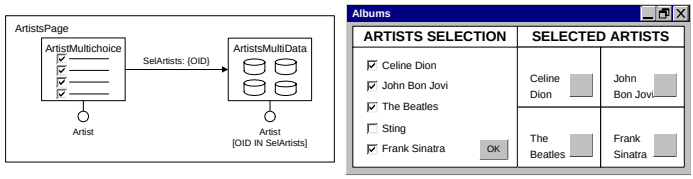


Figure 3.18: -choice index and set-valued link parameter.

Figure 3.17 on the other hand illustrates an example of interpage contextual link. Page `ArtistsPage` contains an index unit, named `AllArtists`, which is defined over entity `Artist`; the index unit is linked to a data unit, named `ArtistDetails`, defined on entity `Artist`, and placed in a distinct page. In this case, the content of the destination unit depends on information provided in the source unit, and the transfer of this context information is associated with the navigation of the link.

Link Parameters and Parametric Selectors

The binding between the source unit and the destination unit of the link is formally represented by a link parameter defined over the link and by a parametric selector defined in the destination unit.

A *link parameter* is a value associated with a link between units, which is transported from the source unit to the destination unit. A *parametric selector* is a unit selector whose condition mentions one or more parameters. An example of these concepts is shown in Figure 3.17, where the link includes a parameter (`CurrArtist`) representing the object identifier of the artist selected from the index, and the data unit has a selector `[OID=CurrArtist]`, which uses the `CurrArtist` parameter to retrieve and display the details of the appropriate artist.

A link parameter can be *single-valued* or *set-valued*; for example, it may hold the set of OIDs of the objects selected from a multichoice index unit. Figure 3.18 gives an example of such a set-valued link parameter associated to a contextual intra-page link.

Source unit	Default link parameters for outgoing links
Data unit	The OID of the object displayed by the unit.
Multidata unit	The set OIDs of the objects displayed by the unit.
Index unit	The OID of the single object selected from the unit.
Hierarchical index unit	The OID of the single object selected from the unit. If the unit has multiple nested entities, the OID refers to an instance of the entity at the top of the hierarchy. Parameters associated with the objects of entities nested at inner levels must be specified explicitly.
Multi-choice index unit	The set of OIDs of the multiple objects selected from the unit.
Scroller unit	The set of OIDs of the selected block of objects, or a single OID if the block factor is 1.

Table 3.1: Default output link parameters of units.

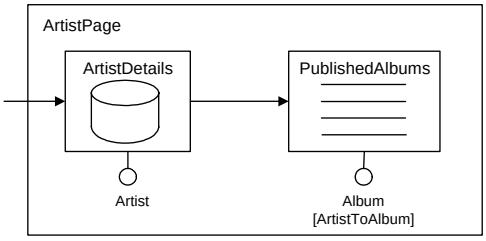


Figure 3.19: Short notation for relationship role selectors.

To make hypertext diagrams more readable, the link parameter specification can be omitted, when the parameters associated with the link are deducible from the context. In this regard, Table 3.1 summarizes the default output parameters of the previously described units, and Table 3.2 reports their default selector conditions.

A very useful application of parametric selectors occurs when one unit must display all the instances of an entity that are related to some instances of another entity. In this case, a selector condition can be specified, which retrieves the subset of the objects of the source entity that are connected by a specific relationship to the object(s) passed in input to the unit by an incoming link. Figure 3.19 illustrates an example of such selector condition, which exploits the relationship `ArtistToAlbum` and, in addition, uses default link parameters.

Destination unit	Default selector
Data unit	OID = < link parameter of type OID of the input link>
Multidata, index, multi-choice index, and scroller unit	OID IN < link parameter of the type OID of the input link>
Hierarchical index unit	The default selector is defined only for the entity at the top of the hierarchy and is: OID IN < link parameter of the type OID of the input link>

Table 3.2: Default selector conditions of units in case of input links without explicit selector conditions.

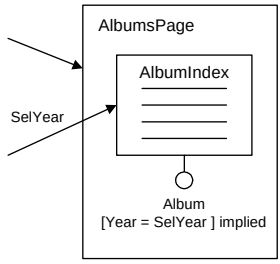


Figure 3.20: Implied selector predicate.

A parametric selector predicate can be tagged *implied*, to denote that the condition expressed by the predicate is optional. In this case, the absence of a value for the parameter used in the predicate can be tolerated, and the selector is evaluated as if the predicate were not specified. For an example, please consider Figure 3.20.

Automatic and Transport Links

When a page is accessed that contains units, which depend on incoming links activated by user-navigated intra-page links or originating in other units of the same page, these units cannot be computed. The solution to this problem is provided by means of an *automatic link*, which is a link that is "navigated" in absence of user interactions, when the page that contains the source unit of the link is accessed, as showed in Figure 3.21. The provided parameter value is heuristically selected (usually the fist available).

All the links seen so far are rendered by means of anchors or confirmation buttons. However, there are cases in which a link is used only

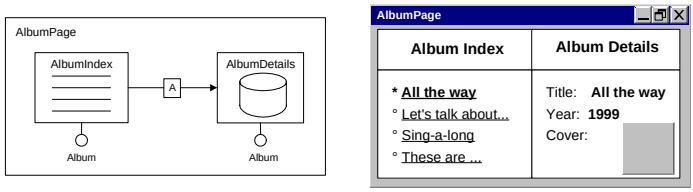


Figure 3.21: Example of an automatic link and its HTML rendition at the first page access.

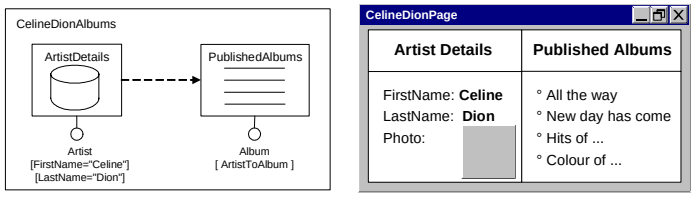


Figure 3.22: Example of transport link.

for passing context information from one unit to another one, and thus is not rendered as an anchor. This type of link is called *transport link*, to highlight the fact that the link enables only parameter passing, and not user navigation. Data units and multidata units, for example, do not allow the selection of objects out of a set, and thus do not require heuristic choices like those made by automatic links. See Figure 3.22 for the graphic notation of transport links.

3.4.5 Global parameters

WebML offers the notion of global parameter to store information that should be available to multiple pages. A *global parameter* is a piece of information, either the OID of an object or a typed value, which can be explicitly set at some point during hypertext navigation and then retrieved to compute the content of some unit later during the navigation. The value of the global parameter is associated with the user's session, so that distinct users may have different values for the same global parameter.

Using a global parameter requires three steps: declaring it, setting a value, and getting the value. The declaration of a global parameter requires the definition of:

- A user-defined *name* for the parameter.
- The *type* of the value stored in the parameter.

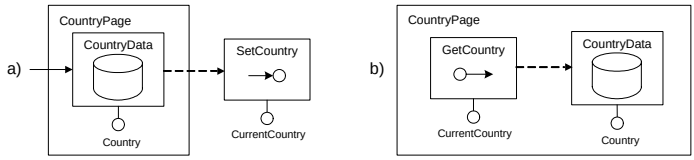


Figure 3.23: a) Setting global parameters. b) Using global parameters.

- A possible *default value*, which is a constant value initially assigned to the parameter.

The value of a parameter is assigned by means of an ad hoc unit, called *set unit* (cf. Figure 3.23(a)). A set unit has only one input link, which is associated with a link parameter holding the value to be assigned to the global parameter. Since the assignment has a global side effect and becomes visible to all the pages of a site view, a set unit is graphically placed in the hypertext diagram outside pages.

A global parameter is retrieved by means of the *get unit* (cf. Figure 3.23(b)), which can be considered the dual operation with respect to the set unit. A get unit has no incoming links, and has only one outgoing link, transporting the value of the retrieved parameter; the unit is placed inside the page where the global parameter value is used, to show the fact that the parameter is retrieved to help the computation of some unit local to the page.

3.5 Content Management Model

For the purpose of content management, a new kind of WebML units, so-called *operation units* or *operations*, are placed outside of pages and linked to other operations or to content units defined inside pages. Differently from content units, operations do not display content, which justifies the fact that they are placed outside pages; rather, they perform actions. Like content units, operations may have a source object (either an entity or a relationship) and selectors, they may receive parameters from their input links, and they may provide values to be used as parameters of their output links. WebML operations obey the following design principles:

- An operation may have *multiple* incoming links, providing values for its input parameters.
- Operations can be linked to form a *sequence*.

- Each operation has one *OK link* and one *KO link*, expressing the operation's outcome. The former is followed when the operation succeeds, the latter when the operation fails.

- An operation may have any number of *outgoing transport links*, which are used to specify link parameters needed by content units or other operation units.

The result of the execution of an operation can be displayed in a page by linking an operation to an appropriate content unit that accepts input parameters from the operation and uses them to retrieve and display the identified information.

3.5.1 Predefined Operations

WebML provides a number of built-in operations, whose meaning is predefined in the language. Due to the orientation toward data-intensive Web applications, most predefined operations address data-management tasks; a few other built-in operations are provided, which offer services of general utility, frequently used in Web applications. They are the *login* and *logout* operations and the *send-mail* operation.

Object creation

The first built-in operation is the *create unit*, which enables the creation of a new entity instance and which we explain in greater detail. Each create unit is characterized by a user-defined name, a source, and a set of assignments, associating values to attribute names.

The input of a create unit is a set of attribute values, typically coming from one input link exiting from an entry unit. The values are used by the create operation to construct a new object; if some attributes have no associated input value, they are set to null, with the exception of the OID that is treated differently: if no value is supplied, a new value, unique with respect to the entity instances, is generated by the operation. The output produced by the create operation is the set of attribute values of the newly created object, including the OID. The output is defined only in case the operation succeeds, and thus can be meaningfully associated as a link parameter only to the OK link, and not to the KO link. The default output of the create unit is the value of the OID attribute, which is assumed as the implicit link parameter of the OK link, if no parameter is specified explicitly.

The example in Figure 3.24 shows the typical usage pattern for create operations, which consists of the combination of an entry unit (`Artist-`

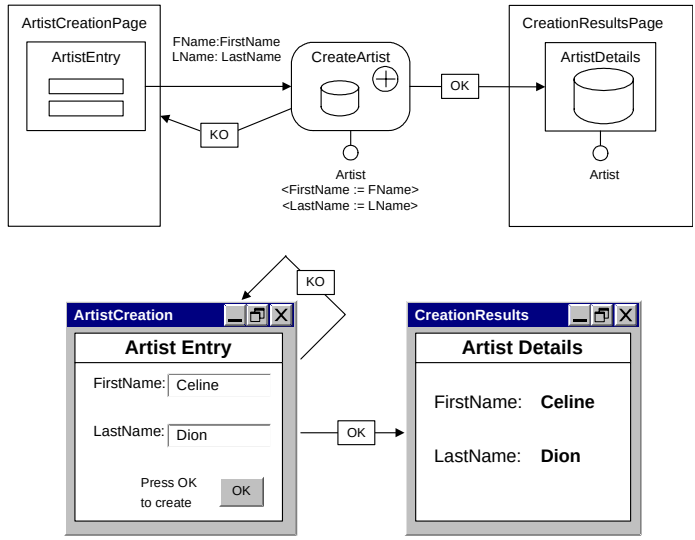


Figure 3.24: WebML graphic notation for create units, and a possible rendition in HTML.

`Entry`) providing input to a create unit (`CreateArtist`), creating a new instance of an entity (`Artist`). In the example, the entry unit has two fields (`FirstName`, `LastName`), to enter the first name and the last name of an artist. The values inserted by the user are associated as explicit parameters with the link from the entry unit to the create operation. These parameters are bound to the attributes of the artist object to be created by means of two assignments, represented below the source entity of the create unit. In the rendition shown in Figure 3.24, the link exiting the entry unit is displayed as a submit button permitting the activation of the operation. The `CreateArtist` operation has two output links: the OK link points to the `ArtistDetails` data unit and is associated with the default link parameter (the OID of the new object); the KO link points back to the `ArtistCreation` page to let the user retry the operation.

Object deletion

The *delete unit* is used to delete one or more objects of a given entity. Each delete unit is characterized by a user-defined name, a source, and a selector.

The user typically chooses at run-time either a single object, displayed by a data unit or selected by an index or scroller unit, or a set of objects, displayed by a multidata unit or selected by a multichoice index unit; the

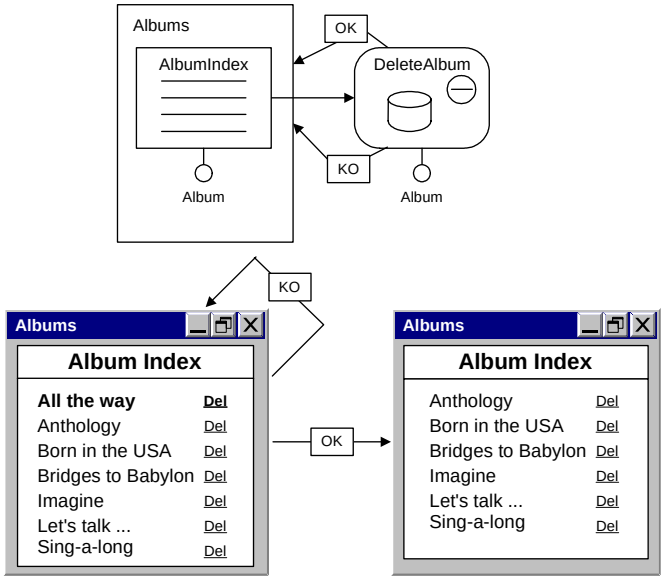


Figure 3.25: WebML graphic notation for delete unit, and rendition in HTML.

corresponding OID or set of OIDs is associated as a link parameter to the incoming link of the delete unit, which actually deletes the objects. Figure 3.25 shows an example usage pattern.

The same configuration would also be valid if set values (provided, for example, by means of a multi-choice index unit) were provided to the delete unit, or if attribute-based selector conditions were specified (for example considering user input by means of an entry unit). The implementation of the delete unit must preserve the referential integrity constraint of relationships.

Object modification

The *modify unit* is used to update one or more objects of a given entity. Each modify unit is characterized by a user-defined name, a source, a selector, and a set of assignments, binding the new values to the attributes of the objects identified through the selector.

A modify unit must be properly linked to other units, to obtain the needed inputs. The OK link of a modify unit is followed when all the objects have been successfully modified. The KO link is followed when at least one of the objects could not be modified, in which case the default outgoing parameter contains the set of OIDs of the objects that could not be modified.

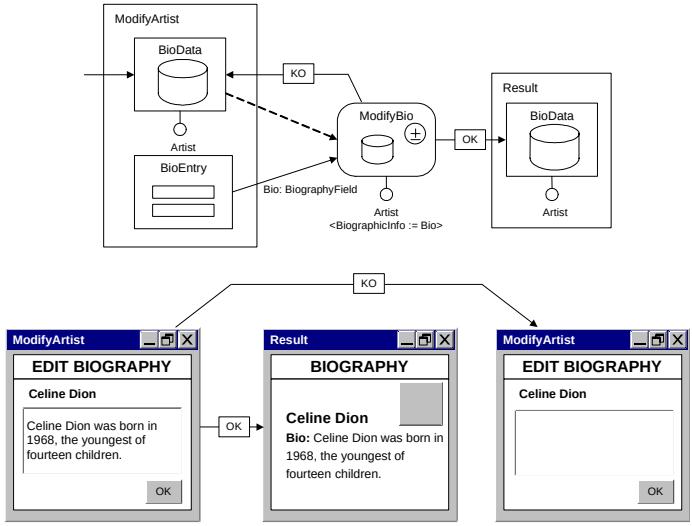


Figure 3.26: Modify unit and rendition in HTML.

The example of Figure 3.26 shows an entry unit used to supply values to a modify unit that allows users to add biographies to the artists already existing in the application's data source.

Relationship creation

A *connect unit* is used to create a new instance of a relationship for each pair of objects of a source and destination entity retrieved by evaluating two selector conditions. The properties of the connect unit are the user-defined name, the source relationship (identifying the two entities to be connected), and two selectors, one for locating the objects of the source entity and one for the objects of the destination entity.

Figure 3.27 shows an example of connect unit attaching a review to an artist. The effect produced by the operation is to connect the selected artist (input parameter `Art`) to the current review (input parameter `Rev`), using the `ArtistToReview` relationship role. Notice the definition of the two selector conditions. The figure reports all parameter names, but since they refer all to object identifiers, they could also be omitted.

Relationship deletion

A *disconnect unit* is used to delete instances of a relationship. As for the connect unit, the properties of the disconnect unit are the user-defined name, the source relationship (identifying the two entities to be

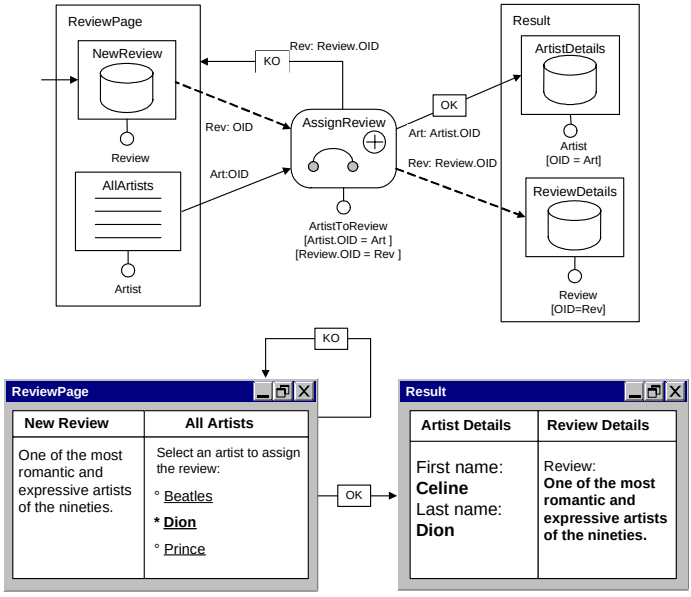


Figure 3.27: Connection unit and rendition in HTML.

disconnected) and two selectors, one for identifying the objects of the source entity and one for the objects of the destination entity.

The operation is used similarly to the connect unit and deletes one instance of the source relationship role for each pair of objects of the source and destination entities identified by the two selectors. Figure 3.28 shows an example where a disconnect unit is used to "detach" one album from an artist.

3.5.2 Access Control and Mail Operations

Login and *logout* operations allow the designer to specify the controlled access to the site, while *e-mail* sending is useful, for instance, for delivering purchase notifications or for sending text or documents to specific recipients.

Login Operation

To implement access control and to verify the identity of a user accessing the site, WebML provides a predefined operation called *login*. The operation has two fixed parameters (`username` and `password`), whose values must be passed in input to the operation by a link, typically exiting from an entry unit, as shown in Figure 3.29.

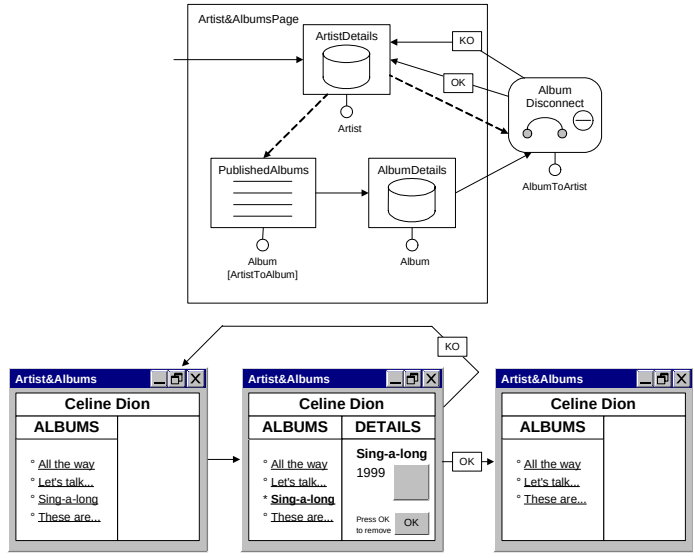


Figure 3.28: Usage of disconnect units and rendition in HTML.

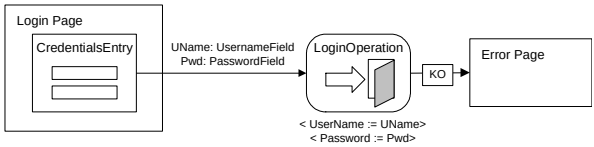


Figure 3.29: WebML login unit preceded by an entry unit for credential input.

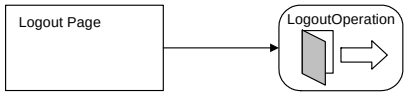


Figure 3.30: WebML logout operation unit invoked via a non-contextual activating link.

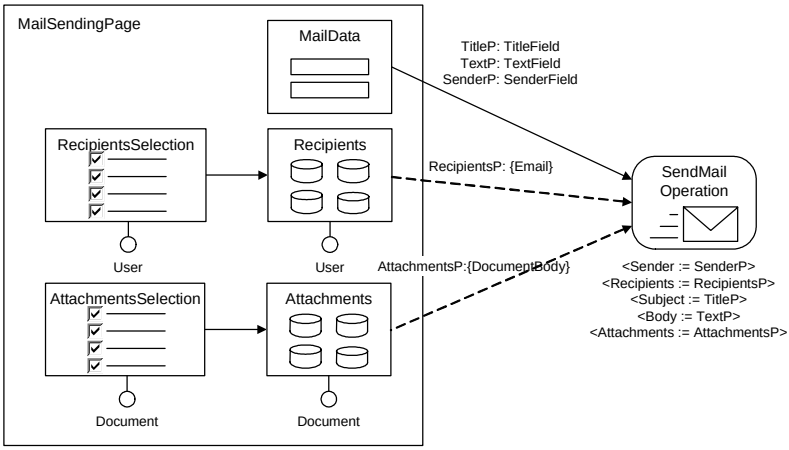


Figure 3.31: WebML sendmail operation unit fed by an entry unit and a data unit.

The login operation checks the validity of the identity of the user, and if the verification succeeds, forwards him/her to a default page. If the credentials are invalid, the login operation forwards the user to the page pointed at by the KO link.

Logout Operation

The *logout* operation is used to leave the session of a logged user and to forward him to a default page with no access control. The logout operation has no input and no output and can be invoked by a simple non-contextual link, as shown in Figure 3.30.

Send-mail Operation

Another predefined WebML operation is the *send-mail* unit, which provides the capability to send e-mail messages. The operation has five parameters: the text of the e-mail, the set of addresses of the receivers, the address of the sender, the subject of the message, and an optional set of attachments. Figure 3.31 shows a typical usage pattern, activating a send-mail operation by means of an entry unit.

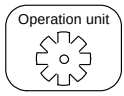


Figure 3.32: Symbol of the WebML generic operation unit.

3.5.3 Generic Operations

In addition to the described built-in operations, WebML allows the designer to define generic operations, such as credit card payments or similar. The generic operation unit has a graphic symbol shown in Figure 3.32.

A generic operation executes outside the WebML context; users may interact with it by supplying input using an arbitrary hypertext pattern, and possibly no longer "come back" to the WebML application. Thus, it is perfectly legal to model an external operation with no output links.

Alternatively, an external operation may have OK and KO links. In this case, upon completion of the execution, the operation is expected to autonomously decide the link to follow, for example based on some result code or exception encountered during processing. In this case, the interaction with the WebML application restarts from the destination page of the followed link.

3.6 Automatic Code Generation

Application development with WebML is assisted by WebRatio [2], a commercial tool for designing and implementing Web applications. The architecture of WebRatio (shown in Figure 3.33) consists of two layers: a design layer, providing functions for the visual editing of specifications, and a runtime layer, implementing the basic services for executing WebML units on top of a standard Web application framework.

The design layer includes a graphical user interface (shown in Figure 3.34) for data and hypertext design, which produces an internal representation in XML of the WebML models. A data mapping module, called Database Synchronizer, maps the entities and relationships of the conceptual data schema to one or more physical data sources, which can be either created by the tool or preexisting. The Database Synchronizer can forward- and reverse-engineer the logical schema of an existing data source, propagate the changes from the conceptual data model to the physical data sources, and vice versa.

A third module (called EasyStyler Presentation Designer) offers functionality for defining the presentation style of the application, allowing

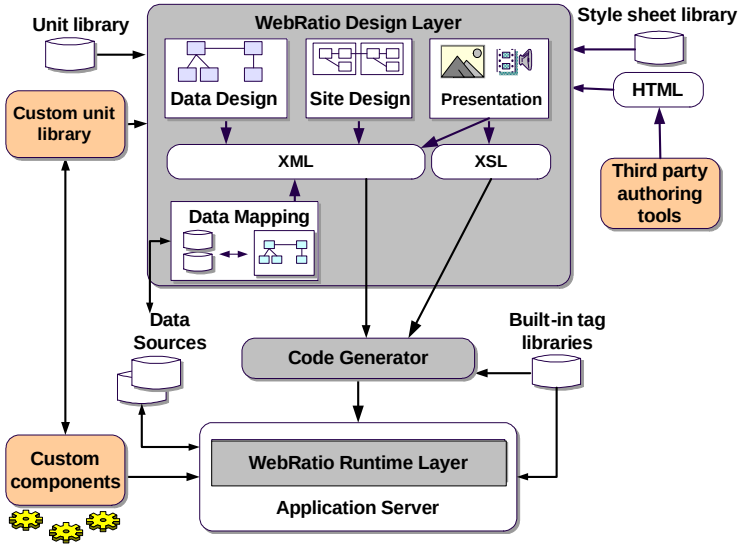


Figure 3.33: The WebRatio Architecture.

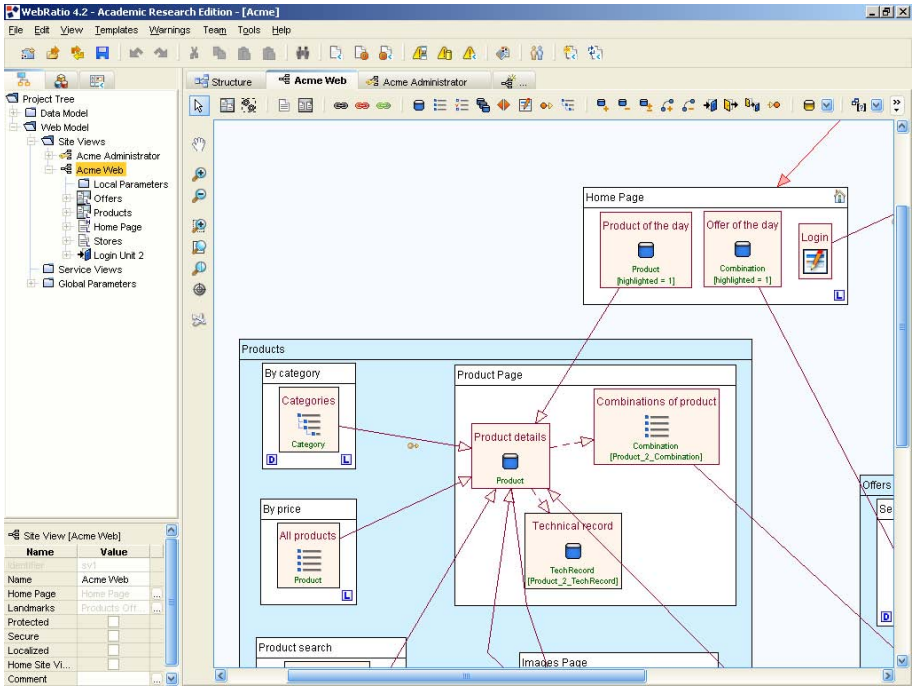


Figure 3.34: The Graphical User Interface of WebRatio.

the designer to create XSL style sheets from XHTML mockups, associate XSL styles with WebML pages, and organize the page layout, by arranging the relative position of content units in each page.

The design layer is connected to the runtime layer by the WebRatio code generator, which exploits XSL transformations to translate the XML specifications visually edited in the design layer into application code executable within the runtime layer, built on top of the Java2EE platform. In particular, a set of XSL translators produce a set of dynamic page templates and unit descriptors, which enable the execution of the application in the runtime layer. A dynamic page template (e.g. a JSP file) expresses the content and mark-up of a page in the mark-up language of choice (e.g. in HTML, WML, etc.). A unit descriptor is an XML file that expresses the dependencies of a WebML unit from the data layer (e.g. the name of the database and the code of the SQL query computing the population of an index unit).

The design layer, code generator, and runtime layer have a plug-in architecture: new software components can be wrapped with XML descriptors and made available to the design layer as custom WebML units, the code generator can be extended with additional XSL rules to produce the code needed for wrapping user-defined components, and the components themselves can be deployed in the runtime application framework.

4 Modeling Context-Aware Web Applications

As we saw in Chapter 2, several kinds of adaptive or context-aware features have been implemented on top of commonly used Web technologies. However, the analysis of the most prominent model-driven design methods for Web applications also showed that the experiences from these applications did not yield corresponding extensions for the conceptual modeling of context-awareness, although almost all of the methods support the specification of advanced personalization features or adaptivity (cf. Section 2.4). In this chapter we propose our own model-driven approach to the design of context-aware Web applications, which, to the best of our knowledge, represents the first conceptualization of such kind of concepts in the domain of the Web. More precisely, we will show how supporting context-awareness in the design process requires to suitably define a reference context model and to augment the expressiveness of the underlying application model, in order to take advantage of context.

New concepts and modeling constructs are formalized as extension of the Web Modeling Language (WebML), where the main goal during the formalization process was to keep the consistency and continuity with respect to the standard version of the language. In particular, the newly introduced concepts and primitives aim to capture those novel requirements that demand for *adaptivity*, i.e. adaptive behaviors during runtime, while *adaptability* is already supported by WebML. In fact, device characteristics and user preferences can be taken into account and used during hypertext design to adapt the application front-end.

4.1 A Conceptual View over Context-Aware Web Applications

Discussing context-awareness in Web applications first of all demands for a precise definition of the term *Web application*, so as to ground the proposed ideas and solutions on a solid technological background. In this work, we concentrate on the "classical" three-tier architecture of

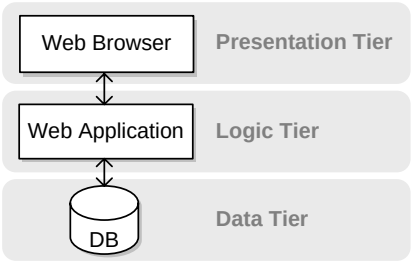


Figure 4.1: Classical three-tier architecture applied to the domain of the Web.

Web applications, consisting of Web browser for *presentation* (thin client with (X)HTML and CSS) and Web application server and database backend for *business logic* and *data* management, as graphically depicted by Figure 4.1.

We therefore assume that the core business logic of the application resides on the server side, and that possible enhanced client-side features (e.g. JavaScript functions or Java applets) only serve presentation purposes. This is in line with the fact that the classical architecture is representative of the most widespread type of Web applications. Accordingly, the proposed modeling approach, at its high level of abstraction, does not depend on any client-side logic.

Note that possible application logic required for the sensing of context properties (on either client or server side) is not considered being part of the Web application to be modeled.

Context-aware applications either *use* context or *adapt* to context to achieve a context-aware feature. The use of context to deliver services or contents occurs in a way that is analogous to the use of standard application data. For instance, displaying context data as application contents is analogous to publishing application data coming from the application's data source. Adaptation to context, on the other hand, may demand for new abilities on the part of the application, i.e. the capability to adapt the application or parts of it to varying context states. In general, context-aware applications may thus also include *adaptability* and/or *adaptivity*, where typically adaptability is based on device characteristics and user profile data and preferences, and adaptivity is based on a dynamic user model.

For the design of Web applications in general it is good practice to adopt a strong separation of concerns between *data* and *hypertext* design (cf. Section 2.4). The approach presented in this dissertation therefore leverages this practice as well for the design of context-aware Web appli-

cations and separates the design and management of context data from the specification of the application's hypertext.

In general, the data that underlie a context-aware application can be characterized as follows:

- *Core application data.* These contain the contents and meta-data used and published by the application to reach its actual application goal. Typically, users are interested in application data.
- *User profile data.* These are necessary for two main reasons: (i) *adaptability* with respect to the profile, and (ii) *access control* with respect to the user's identity. Access control (i.e. the identification and authentication of users) is a main feature of most modern software systems and represents a specific aspect of adaptability. Once a user's access rights are known, an application typically allows the user only access to those resources (e.g. contents and services) that are compatible with the user's role. Contents and services are then adapted and presented according to his/her profile and device characteristics.
- *Dynamic user model data.* These express a user's dynamic profile data, i.e. data that may change during the runtime of the application, such as preferences over application contents (e.g. the preferred author of a user) or ownership relations between a user and contents (e.g. the ownership of a comment on a blog). Data in the dynamic user model is typically taken into account by applications through *adaptivity*.
- *Context data.* These data describe the context of the interaction between the user and the application; typically context data change rapidly (compared to, for example, the user model). This property implies the need for *adaptive* features that are able to perform application adaptations during runtime also according to context data.

As in this dissertation we concentrate on the development of context-aware Web applications, we forget for a moment about the peculiarities of user and personalization data and consider them as part of the general application data (we will turn back to this distinction in the next section), so as to better focus on the characteristics of context data.

Figure 4.2 proposes a functional architecture that takes into account the separation of data and hypertext design and highlights some issues related to context data acquisition and the flow of context data within a context-aware application.

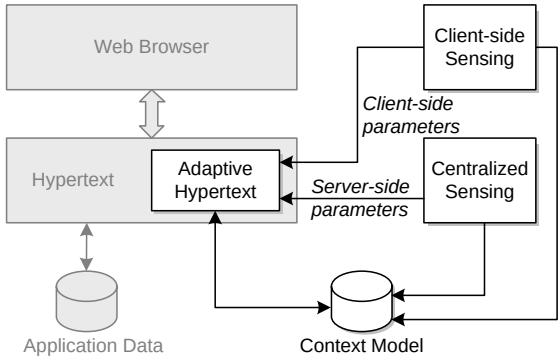


Figure 4.2: Context data in context-aware Web applications. Gray shaded boxes correspond to conventional, non-adaptive parts, white boxes correspond to extensions required to support context-awareness.

The application's data source includes both the *application data* (i.e. the business objects that characterize the application domain, the user and the preference data) and the *context model*, which offers at any moment an up-to-date representation of the context state. The context model captures all the context-characterizing properties (i.e. attributes and changes in time) that enable the system's adaptation to the context.

In order to feed the context model with data and to update it on changes, context data needs to be *sensed* and *communicated* to the Web server that hosts the application. In our interpretation of context-aware Web applications, there are three main communication mechanisms that allow the sensing devices to pass context data to the application: (i) as parameters sensed at the client side and sent to the application (i.e. to the adaptive hypertext, which is able to interpret the incoming data); (ii) as server-side parameters directly filled by the centralized sensing infrastructure (i.e. HTTP session variables, immediately available to the application for consumption/evaluation); and (iii) by means of direct updates of the context model. Typically, client-side parameters are generated by the client-side sensing tools, server-side parameters are filled by the centralized sensing tools, and database updates may be performed by both sensing solutions.

An application may then consist of adaptive (i.e. context-aware) and non-adaptive parts; we call the former *adaptive hypertext* in Figure 4.2. With the term adaptive hypertext we indicate all those pages of a Web application that present some form of adaptive behavior, while the non-adaptive hypertext collects all those pages that do not present any adap-

tive behavior. In a context-aware application, for adaptation purposes, the adaptive hypertext also exploits context data during the rendering of the hypertext pages. Non-adaptive Web application components are shaded in gray in the figure, while components supporting context-awareness are highlighted in white.

Hence, context-awareness in Web applications, according to the proposed architecture, requires the following tasks to be addressed:

1. *Context model definition and representation* in an application-accessible format, as described in Subsection 2.2.3. In this task, the main context properties required for supporting the application's adaptivity requirements need to be identified and modeled. The modeling typically comprises both physical and logical context (cf. Subsection 2.2.2).

2. *Context model management*, consisting of:

   a) *Context data acquisition* by means of measures of real-world, physical context properties, characterizing the usage environment. Measures may be performed at client side or by means of dedicated, centralized sensing infrastructures. The so acquired data serve to update the context state maintained in the context model.

   b) *Context data monitoring* to detect those variations in context data that trigger adaptivity. Any variation may cause an automatic (*context-triggered*), adaptive behavior of the Web application. Context monitoring thus typically implies translating physical context into logical context and deciding whether detected changes demand for adaptation or not.

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

While the definition of the context model and the monitoring of context data can easily be assisted by proper context modeling methods and a proper runtime framework providing basic monitoring facilities, it is not as easy to assist designers in the development of a suitable context acquisition (i.e. sensing) infrastructure. In fact, the former two activities can be generalized to a useful degree, while the design of the sensing infrastructure results to be tightly coupled with the specific application

requirements and the resulting technological design choices, affecting both hardware and software artifacts. For this reason, an exhaustive discussion of sensing technologies would be out of the scope of this work. Instead, we focus our investigation on the way context data may be *communicated* to a context-aware Web application, as graphically depicted in Figure 4.2.

Context monitoring may be performed at each user-generated *navigation action* that activates the application logic of the Web application and enables the application to adapt, if required[1]. Due to the dynamic nature of context, a better reactive behavior could be achieved by *periodically communicating* fresh context data and recomputing pages; if the evaluation of the context state demanded for adaptation, proper adaptivity actions might be triggered automatically, even in absence of user interactions. Instead of polling adaptivity, the best solution would be to have an *active monitoring* mechanism operating autonomously and transparently in the background and independently from the user's interactions, in order to trigger adaptivity in real time. Depending on the individual implementation of each application, one of the previous methods to enable the monitoring of context data can be adopted.

Hypertext adaptation, finally, must be performed during the dynamic computation of the hypertext interface of the application. Ideally, adaptations would need to be performed immediately in response to the detection of a situation demanding for adaptation, but in the case of Web applications, due to the lack of push mechanisms in traditional Web technologies, this can only happen when computing a page. In Subsection 4.3.5 we will discuss in more detail *when* adaptations can be performed.

As for the object of the adaptation, Brusilovsky [28] already identified the components that can be adapted in adaptive hypermedia: presentation and navigation. Analogously, in context-aware Web applications, *adaptive behaviors* can be applied to:

1. *Contents and services* delivered by accessed pages, which can be adapted on the basis of the current context state.

2. The *navigation*, in form of automatic navigation actions toward pages of the same application, which are better suited to the effective context conditions.

3. The whole *hypertext structure* to support coarse-grained runtime adaptations (e.g. of the layout of the application), for example

[1] This solution is for example adopted in most of the adaptive applications and modeling solutions described in Section 2.4.

due to changes of the user's device, role, or activity within a multi-channel, mobile environment.

4. *Presentation properties* , in order to provide more fine-grained adjustments of the application's appearance (e.g. referring to style properties).

In the following sections, we will illustrate how the previous requirements, from context data representation and acquisition to the specification and enactment of adaptivity actions, can be expressed in the model-driven design language WebML. According to the WebML design method, supporting context-awareness occurs along three dimensions: data design, hypertext design and implementation.

4.2 Modeling Context for Adaptivity

As outlined in Section 2.2, even though there are several properties commonly regarded as *context attributes* (e.g. position, time, or device characteristics), there exists no universal context model that applies to all kinds of applications. For this reason, also in the context of WebML we can not prescribe any precise, rigid context model for WebML applications; we rather introduce some WebML-specific modeling guidelines that enable the designer to provide context-aware applications with suitable context meta-data.

Context data can derive from several sources integrating sensed, user-supplied and derived information [60, 61]. While user-supplied data are generally reliable and tend to be static, sensed data are highly dynamic and can be *unreliable* due to noise and sensor errors. The problem of unreliability has been addressed in literature, for example, by associating context information with quality data [62]. Although we recognize the importance of reliable context data, in this work we rather concentrate on the exploitation of context in the design of Web applications. For simplicity, throughout this dissertation we thus consider sensed data as trustworthy.

As to the *dynamics* of context data, it is worth to note that modeling context as relational data does not allow designers to explicitly represent the dynamic nature of context in the context model. The dynamics of context data, however, does not influence the design of adaptive WebML hypertext schemas, as will be shown in Section 4.3; what is required, instead, is a proper context monitoring mechanism that allows the application to trigger adaptivity in function of the dynamics of its context. This problem is dealt with in Section 5.3.

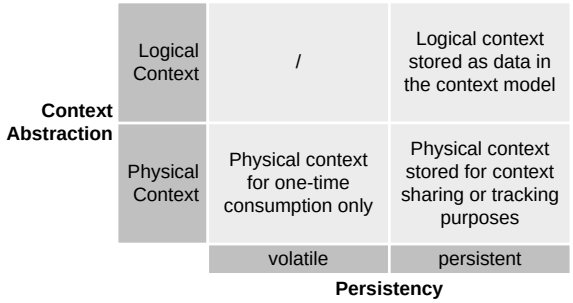


Figure 4.3: Persistence of physical and logical context data.

4.2.1 Characterizing Context Data

The main goal of context modeling is the formalization and abstraction of the context properties that affect the application. In this regard, a first characteristic distinguishing context properties was already introduced in Subsection 2.2.2, i.e. the distinction between physical and logical context. We call *physical* context those properties that are immediate representations (e.g. the values of an analog/digital converter) of sensed, physical quantities, and *logical* context those properties that enrich physical context with semantics and additional abstractions of the raw sensed data (e.g. the city corresponding to physical longitude and latitude values).

A second characteristic affecting the topology of the context model is the *persistence* of context properties in the system, i.e. the decision whether individual context properties represent *persistent* data or *volatile* data. Persistent data need to be stored in the application's data source and therefore require proper data entities being modeled as part of the context model, while volatile data do not need any storage and can thus be omitted from the context model. The context communication mechanisms described in Figure 4.2 enable the communication of both persistent data (i.e. via direct database updates) and volatile data (i.e. via client-side or server-side parameters).

Starting from these two characteristics, Figure 4.3 summarizes the different kinds of context data that influence the specification of the context model:

- *Volatile physical context.* Context communicated via client-side parameters or via server-side session parameters represent volatile data. They are immediately available during the execution of the application, independently from the underlying context model. Volatile context data do not need to be enclosed in the context

model; they might be used during runtime to query logical context data.

- *Persistent physical context.* Context sharing (e.g. between members of a same group) or tracking (e.g. to derive differential context properties or to keep a context history) typically require the persistent storage of data. Persistent physical context data thus need to be included into the context model and updated according the their dynamics.

- *Persistent logical context.* Logical context is stored as data in the context model, so as to enable the data-driven transformation of physical context into logical context. Logical context is typically static; dynamic updates and/or extensions can, however, be supported as well.

Physical and logical context data coexist in the application's data source. This coexistence typically requires a transformation or mapping that translates raw data into information that can directly be used when specifying hypertext schemas to be rendered to users. In line with the data-driven approach that characterizes WebML, we propose a formalization of such transformation at data level by means of suitable associations representing the mapping. As typically all logical context data are thus persistently stored in the context model, we do not expect the use of *volatile logical context.*[2]

Summarizing, we can say that the definition of an application's context model is influenced by the persistence requirements the application poses to the context properties: persistent context data needs to be modeled at data level; volatile data do not need to be represented as information objects. They however need to be captured when modeling the behavior of the adaptive hypertext.

4.2.2 Modeling User, System and Environment Data

According to our definition of context given in Section 1.4 (Definition 1), context data can be partitioned into *user*, *system*, and *environment* context data:

[2]Remember that all context data is kept at the server side. But also in the case of context data managed at the client-side (e.g. by means of AJAX scripts), logical context data need to be explicitly provided and persistently stored by application developers. Moving application logic from the server to the client does not imply volatile logical context data.

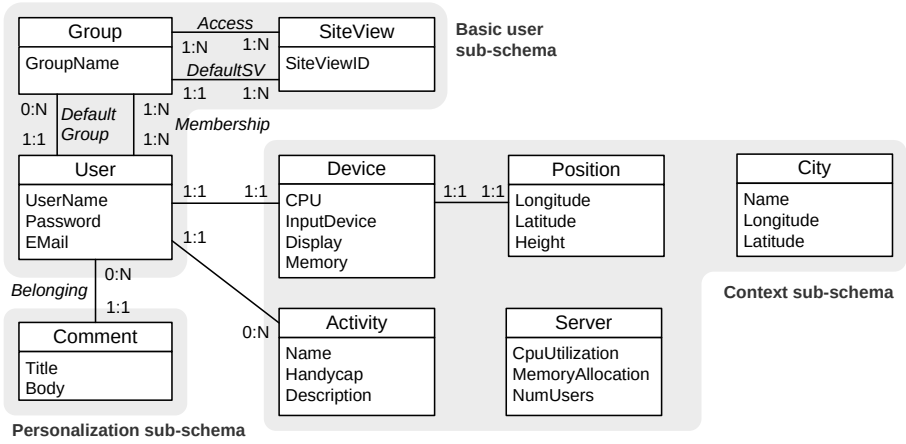


Figure 4.4: Adaptation-triggering data in WebML applications, partitioned into basic user sub-schema, personalization sub-schema and context sub-schema.

- *User context data* refer to those context properties that are directly related to the identity of the user, e.g. the position.
- *System context data* contain (runtime) properties about the context-aware system, e.g. the current workload.
- *Environment context data* express properties about the environment that hosts the interaction between the user and the application, e.g. weather conditions or time.

While system and environment context data are *shared* among all users of the application, user context data are *individual*. As a consequence, user context data need to be considered in relation to individual users. When modeling context data, this means that the data entities for the individual context must be connected (directly or indirectly through other relationships) to the user, so that during application execution individual context data can be retrieved starting from the identity of the user.

4.2.3 Example Data Schema for Adaptation in WebML

Figure 4.4 illustrates an example Entity-Relationship diagram with user profile, personalization, and context data, as it could be adopted in WebML. Each kind of data is modeled by means of a proper ER sub-schema:

- *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` associates access rights to users (i.e. a role), and the entity `Site View` contains the site views that may be accessed by the members of a group. Site views are WebML hypertext schemas (i.e. 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 is allowed to access. This relationship is required only in the case of context-aware applications that may require different interaction and navigation structures for a same group, according to varying context conditions. Therefore, depending on the context state, the application is able to determine the most appropriate site view and to forward the user accordingly.

- *Personalization sub-schema.* Personalization in WebML is achieved by specifying relationships between entities of the application schema and the entity `User`. For instance, the entity `Comment` is connected to the entity `User` by means of the relationship `Belonging`, which expresses the fact that comments belong to individual users. 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.

- *Context model sub-schema.* Finally, Figure 4.4 proposes a possible configuration of context meta-data, as it could apply for example to mobile and multi-channel Web applications.
The entities `Device` and `Activity` represent explicitly provided logical context data, and the entities `Position` and `Server` represent sensed/measured physical context data, while the entity `City` represents logical context data translating a physical position into the logical concept city. Note how the user context entities `Device`, `Position`, and `Activity` are associated to the entity `User`, while the system context entity `Server` is not associated to any other entity. The individual city of a user can be derived at runtime by querying the entity `City`.

In a certain sense, also the relationship `Access` in the user profile sub-schema could be considered part of the context sub-schema, as in WebML multiple site views per group are only justified if context conditions require the overall hypertext structure to adapt according to the context. Users belonging to a given group might in fact operate in totally different contexts (e.g. expressed by the entity `Activity`) and might hence require different site views with different hypertext structures.

4.3 Modeling Adaptive Hypertexts

While the first step of the WebML design method, i.e. data modeling, does not require any extension of the modeling primitives for capturing context data (the standard Entity-Relationship primitives suffice), hypertext modeling does require proper model extensions to cope with adaptivity. In this section we therefore introduce the new concepts and primitives that have been developed at the hypertext level to express the desired adaptive behaviors. Also, we clarify how different adaptivity policies can be used to enact the adaptation of the application and how the new paradigm impacts on the computation of hypertext pages.

4.3.1 Context-Aware Pages

Our basic assumption in the modeling of context-aware hypertexts is that context-awareness or adaptivity is a property to be associated only to some *pages* of an application (the *adaptive hypertext*), not necessarily to the application as a whole. Location-aware applications, for example, adapt "core" contents to the position of a user, and "access pages" (including links to the main application areas) typically are not affected by the context of use.

As can be seen in Figure 4.5, we tag context-aware or adaptive pages with a C-label (standing for *context-aware*) to distinguish them from conventional pages. The label indicates that an adaptivity logic is associated with the page, and that during the execution of the application this logic must be taken into account when computing the page. The associated adaptivity logic may serve for adapting the page content or for modifying the predefined navigation flow.

4.3.2 Context Clouds

We call the adaptivity logic (i.e. the set of adaptivity actions attached to a page) *context cloud*. As sketched in Figure 4.5, the cloud is external to the page, and the chain of adaptivity actions it clusters is kept separate

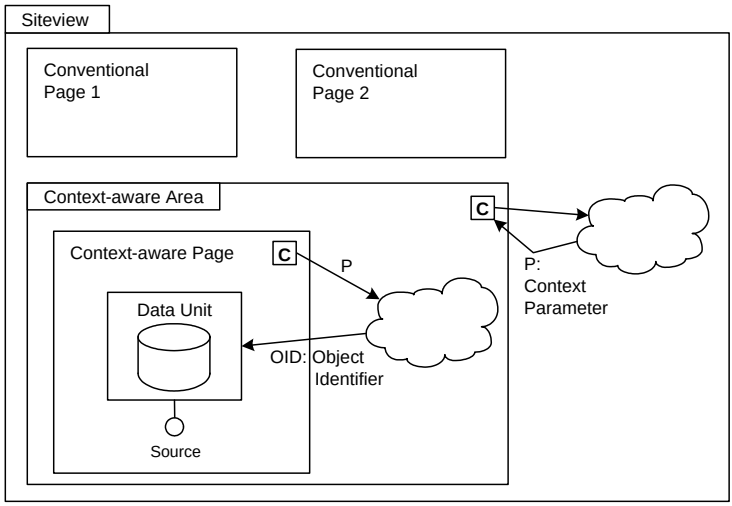


Figure 4.5: WebML hypertext schema with two conventional pages, one context-aware page, and one context-aware area, together with their context clouds. The parameter `P` exemplifies the propagation of reusable context data by hierarchically passing context parameters from an outer area to an inner page.

from 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 in the context cloud that might be required to perform the adaptation of page contents to new context conditions.

Adaptation actions specified in the context cloud of a C-page typically present effects that are visible in the page they are attached to. The notion of context-aware pages and context clouds therefore defines what we call a *localized* adaptivity rule:

Definition 9 (Localized Adaptivity Rule) *The scope of a* localized *adaptivity rule is strictly coupled with a fixed set of hypertext pages, where* scope *refers to those (adaptive) pages to which the page's adaptivity actions are associated.*

However, there might also be the need for adaptation actions with effects that are spread over multiple pages. For this purpose, we exploit the hierarchical structure of hypertexts.[3]

4.3.3 Structuring Context-Aware Hypertexts

The central context-aware hypertext element is the page. However, as represented in Figure 4.5, we also propose to define context-aware *containers* (*site views* and *areas*, in terms of WebML) as grouping facilities. This allows the designer to insulate and to specify only once adaptivity actions that are common to multiple C-pages in a container, and thus to reduce the redundancy of the schema. There are in fact actions to be evaluated for every C-page, and associating such actions to a page container allows designers to keep the specification clean and easy to read. Context clouds associated to containers and pages are evaluated recursively, starting from the outermost container and ending with the cloud associated to the page. The notion of context-aware container allows us to define *sparse* adaptivity rules:

Definition 10 (Sparse Adaptivity Rule) *We talk about* sparse *adaptivity rules in those cases, where adaptivity actions are associated to containers that contain multiple pages; the scope of such actions spans over a set of pages.*

Figure 4.5 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 might happen in the case of context parameters, it passes such values back to the C-label that activated the computation of the cloud. Subsequently, such parameters can then be "consumed" in the context clouds of the inner levels. As for context-aware pages, parameter passing from a container to its context cloud occurs through the cloud-activating link. Links exiting from the last evaluated cloud, i.e. at the end of the last adaptivity action chain, 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, e.g. if the data are to be shared or a history is to be tracked. Hence,

[3] We propose to structure adaptivity rules according to the hypertext structure of the application. In AHAM [63], for example, the authors instead propose to structure "pedagogical rules" in function of the hierarchical structure of *concept components* in the *domain model* of an adaptive hypermedia application.

if persistent context data are adopted, we propose two levels for the specification of adaptivity actions:

- *Actions for context model management*, addressing operations for context data acquisition and the consequent context model updating, should be associated with outer containers (site views or areas) and are inherited by inner containers (areas or pages). These adaptivity actions need to be executed prior to the execution of any other action possibly specified in an inner context cloud, as such "internal" actions could depend on persistent context data acquired and stored in the data source through "external" actions.

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

4.3.4 Enabling Adaptivity: Context Monitoring

In order to manifest context-aware behaviors, C-pages must be provided with the capability to monitor the context state and to trigger their adaptivity actions, if required.

The standard HTTP protocol underlying most of today's Web applications implements a strict *pull* paradigm, in which refreshes can only occur in response to client-side generated page requests. Therefore, in the classical Web architecture, lacking proper push mechanisms, context monitoring can occur only when a page is computed, i.e. when a respective page request has reached the Web server. Three main solutions can be adopted to trigger the evaluation of context clouds: (i) context evaluation on user-generated page requests, (ii) periodical, automatic refreshes of viewed pages to enable context evaluation, and (iii) active context evaluation to trigger context clouds in real time. The first solution is not able to cope with the dynamic nature of context. The *periodic refresh* of context-aware pages provides a way to ensure the update of the page even in absence of explicit user actions enabling the re-computation of the page. In Section 5.3 we will also show an active mechanism for triggering adaptivity, which operates independently from the user in the background; however, this does not alter the modeling solution proposed in this chapter.

In absence of dedicated server-side *push* mechanisms for delivering updated pages, the HTML `http-equiv META`-option, or also JavaScript, JavaApplets, or Flash scripts, provide valuable client-side mechanisms to "simulate" the required active behavior. More precisely, this simula-

tion implies generating periodic HTTP requests toward the application server, which may serve a twofold purpose:

- On one hand, they provide the necessary polling mechanism to query the context model and trigger the context cloud attached to the page to be refreshed, thus reacting to possible context changes.
- On the other hand, generating page requests may enable the client to transmit client-side sensed data, thus enabling the communication of context data to the application server.

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

Page Context

In general, the *state* of the context is expressed by the values of all the persistent parameters stored in the context model and of the volatile parameters sensed at the client or server side. However, an individual page's adaptive behavior is typically influenced by only a subset of the overall context data or, more specifically, by a function expressed over context data, not just by the simple values and/or dynamics of an application's context parameters. This context data subset thus corresponds to a page-specific view over the application's context data, narrowing the focus of the context monitoring activity. This observation leads to the definition of a new concept, i.e. *Page context*, which can be leveraged to enhance the efficiency of the context monitoring activity.

Definition 11 (Page Context) *The* Page context *of a page corresponds to a page-specific view over the application's context data, capturing all (and only) those context characteristics that effectively determine the adaptive behavior of the page.*

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

4.3.5 Adaptivity Policies

Starting from the continuous (in our case periodical) monitoring of the context state, it is possible to define two different *adaptivity policies* for context-aware pages, assigning different priorities to users and context:

- *Deferred Adaptivity*: the *user* is granted the highest priority. Therefore, after the user has entered the page and the page has been rendered according to the user's selections, adaptivity only starts after the first refresh interval of the page.
- *Immediate Adaptivity*: *context* is granted the highest priority. Adaptivity actions are therefore evaluated every time the page is accessed, prior to the actual page computation. This means that the page is subject to adaptation each time it is rendered, even at the first time the page is accessed by the user.

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

Note that in addition to these adaptivity policies, we recognize that there may be situations that demand for an explicit control of the adaptation dynamics by the user. Therefore, should for example a user temporarily not be interested in having the contents adapted to his/her location, he/she can simply disable/enable adaptivity at will.

Adaptivity policies can also be associated to context-aware containers. When a C-page is requested, also the possible context clouds of its containers are evaluated recursively (from the outermost one to the innermost one), according to the adaptivity policy associated to each container. In general, a container's adaptivity policy is independent from the policy of inner containers and pages[4]. Therefore, it may happen that the

[4]The only exception occurs when a dependency exists among clouds at different levels, i.e. due to parameter passing. This situation must be taken into account

actions in a container's context cloud are evaluated immediately, even if the actions associated to inner containers or pages adopt a deferred evaluation, or vice-versa. If, for example, the adaptivity actions associated to the container serve for tracking a context history, they could require an immediate policy, while inner adaptivity actions keep their deferred policy for front-end adaptations. The hierarchical definition of context clouds may therefore also be considered a facility to achieve different "layers" of adaptivity actions.

In WebML, the adaptivity policy for context-aware pages and containers is declared as a property of context-aware pages and containers by means of the `Adaptivity_Policy` property.

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 the user's selections; only afterward that page might become subject to adaptation, according to the new 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. We therefore, in general, recommend the selection of the adaptivity policy that is appropriate to the application goals and that is able to minimize the application behaviors that could be perceived as invasive or annoying by the users. In order to choose the right adaptivity policy for an adaptive page, a developer therefore needs to predict what kind of adaptive behavior a user will expect when accessing that page.

4.3.6 Specifying Adaptivity Actions

The main novelties for modeling context-aware pages lay in the specification of suitable adaptivity actions clustered into context clouds. Therefore, in the following we will introduce some new WebML modeling concepts that ensure full coverage for the specification of context model management and hypertext adaptation. The new primitives allow designers to visually specify actions for acquiring and updating context data and to define proper adaptivity rules.

by designers when associating individual policies to containers and pages.

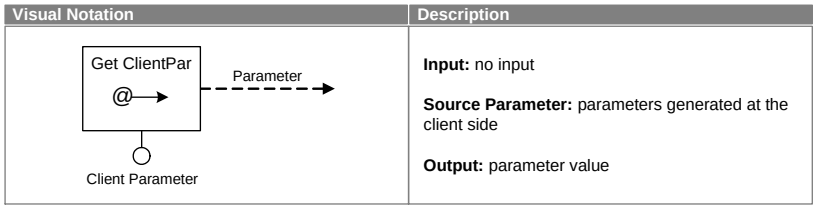


Figure 4.6: Visual notation of the *Get ClientParameter* unit.

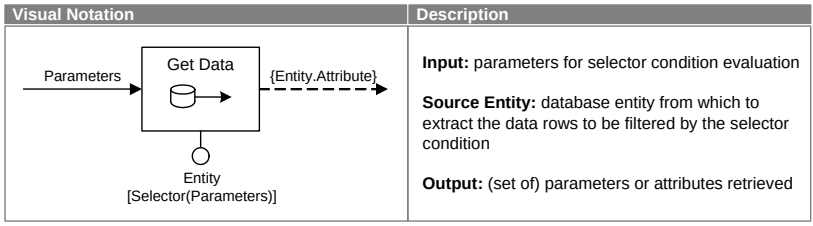


Figure 4.7: Visual notation of the *Get Data* unit.

Managing Context Data

In order to support adaptivity with respect to the current context state, the application must be able to acquire and manage context data according to the mechanisms illustrated in Section 4.1. For this purpose, some new WebML operations have been defined, which, together with the already available operations, provide the necessary primitives for:

- *Specifying the acquisition of fresh context data through client-side parameters.* A new `Get ClientParameter` unit (see Figure 4.6) has been defined to support the retrieval of parameters generated at the client side and communicated back to the application via client-side parameters (e.g. parameter-value pairs attached to the page request query string).

- *Specifying the acquisition of fresh context data through server-side parameters.* Context data directly made available as HTTP session parameters can be accessed by means of conventional WebML `Get` units (see Subsection 3.4.5).

- *Specifying the acquisition of context data from the context model.* In conventional applications, page computation implies retrieving data to be published in pages from the data source. The execution of adaptivity actions may also require the retrieval and evaluation of context meta-data. For this purpose, a so-called `Get Data`

unit (see Figure 4.7) has been introduced, enabling the retrieval of values (both scalars and sets) from the data source according to a selector condition. The semantics of the `Get Data` unit is similar to the one of content publishing units, with the only difference that data retrieved from the data source are not published in hypertexts, but just used as input to subsequent units or operations. It therefore provides a means to access 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 can be used to update the context model at data level. This action consists in modifying values previously stored in the data source. In WebML, this is already facilitated by operation units (see Section 3.5) providing support for the most common database management operations (`modify`, `insert`, `delete`).

Evaluating Conditions

The execution of adaptivity actions may be subject to the evaluation of some *conditions*, refining the triggering logic for context clouds. The most recurrent pattern consists in evaluating whether context changes demand for adaptation.

The evaluation of conditions is specified by means of two control structures, represented by the `If` and `Switch` operation units, which have been introduced for workflow modeling in WebML [64].

Executing Adaptivity Actions

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

- *Adapting Page Contents.* Page contents are adapted by means of proper data selectors, whose definition is based on context parameters retrieved from the context model or newly computed within the page's context cloud. The use of parameterized selectors allows for both *filtering* data items with respect to the current context and conditionally *including/excluding* (i.e. showing/hiding) individual content units.

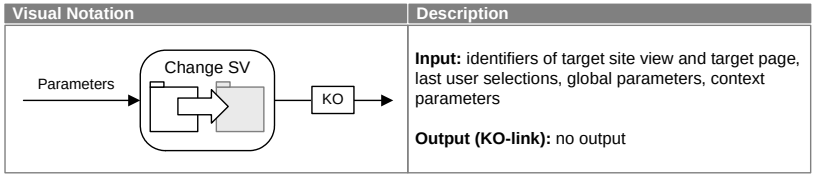


Figure 4.8: Visual notation of the *Change SiteView* unit.

- *Adapting Navigation.* In some cases, the effect of condition evaluation within the context cloud can be an automatic, i.e. context-triggered, navigation action, causing the redirection of the user to a different page.
  The specification of context-triggered navigations just requires connecting one of the links exiting the context cloud to an arbitrary destination page of the hypertext. Therefore, links exiting the context cloud and directed to other pages than the context cloud's source page represent automatic navigation actions.

- *Adapting the Site View.* In some cases, a context-triggered switch toward a different site view may be required. Changes in the interaction context may in fact ask for a coarse-grained restructuring of the whole hypertext, for example because the user device has changed, or because the user shifted to a different activity.
  To switch between different site views, we have introduced a `Change Site View` unit (see Figure 4.8), which takes in input the identifiers of the target site view and the target page, to be visualized in case a switch toward the specified site view is required. In order to support "contextual" switching, the input link also transports parameters characterizing the current state of interaction, i.e.:

  1. The input parameters of the source page, which represent the last selections operated by the user.
  2. Global parameters, representing session data (e.g. user OID and group OID), as well as past user selections that have been used for the computation of the current page.
  3. Client-side and server-side context parameters retrieved during the latest performed data acquisition cycle and characterizing the current context state.

- *Adapting Presentation Style.* Sometimes context changes may require only fine-grained adaptations of presentation properties (e.g.

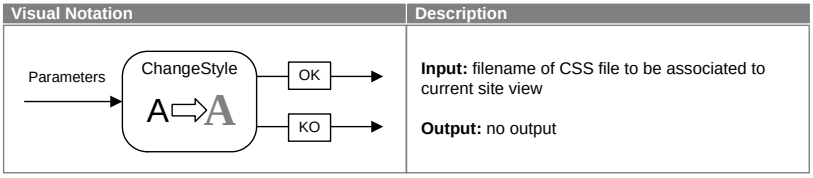


Figure 4.9: Visual notation for the *Change Style* unit.

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 the `Change SiteView` unit or by conditionally including/excluding content units, we have also defined a `Change Style` unit for dynamically assigning presentation style properties (see Figure 4.9). Style properties are collected in proper `.css` (Cascaded Style Sheet) files, and the unit enables the application to change its associated style sheet at runtime.

4.4 Computation of Adaptive Hypertexts

In addition to the extensions of the WebML hypertext model, context-awareness in WebML also demands for a revision of the WebML page computation algorithm [1]. Indeed, in the case of context-aware pages, the page computation logic also 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 not only tackle the problem of how to compute the units contained in a page, but it also must guarantee the correct activation

and execution of context clouds. As already expressed by Figure 4.5, this implies recursively evaluating each context cloud before the 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 (stored in the `newPage` variable) that, when the evaluation of the context cloud triggers an automatic navigation toward a different page, represents a pointer to the target page. This value is `null` if no automatic navigation is required. 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 or the C-page under evaluation.

4.4.1 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 at a time has to be considered, the computation of the unit results to be ambiguous. Some *specificity rules* are therefore necessary to decide which of the incoming values to use.

For ordinary pages in WebML, the specificity of input parameters is assessed according to the following principles [1]:

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 choices or which 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 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.

Coherently 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 first page access through the user.

Figure 4.10 illustrates the steps required at runtime for computing dynamic page templates, and highlights the additional computation steps required in the case of context-aware pages.

To identify that a page has been requested by the automatic refresh mechanism and not by the user, a proper parameter (i.e. `automatic`) can be appended to the HTTP page request in case of automatic requests. As the figure shows, when the Web server receives a new page request, it decodes the incoming request parameters and, only for C-pages, 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:

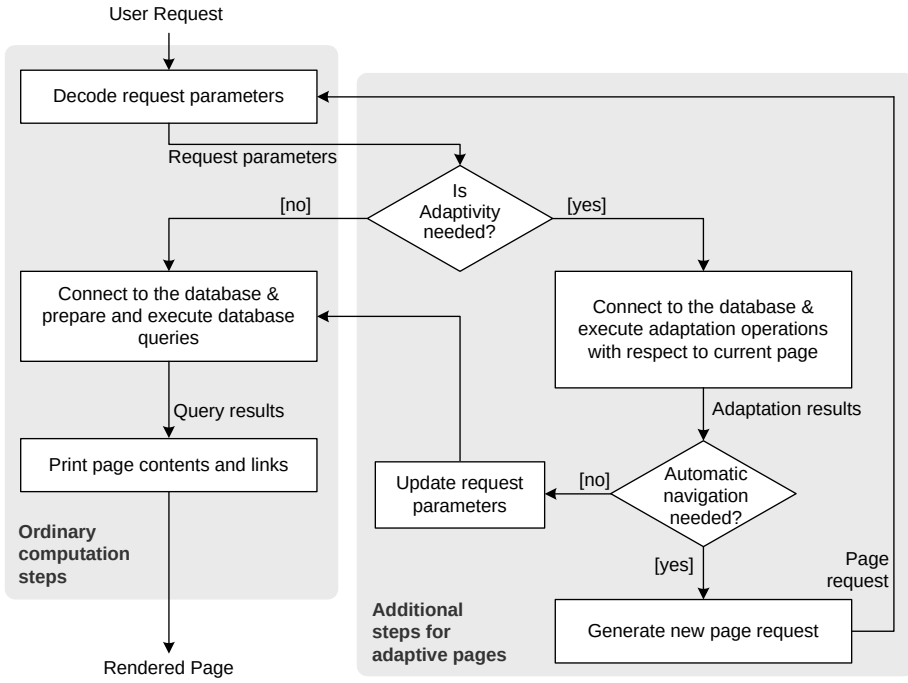


Figure 4.10: Computation of context-aware page templates.

- If adaptivity is not required (i.e. for conventional pages or for the first user access to a C-page 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:
  - *Page adaptation*: the database is accessed to read the current state of the context, and request parameters are updated accordingly. Thus, computation proceeds as for ordinary pages. The new values of the page parameters will cause the adaptation of the page.
  - *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` set to `"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., [65, 66, 67]) 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 to check 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 [65].

A design guideline to prevent 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 the next adaptivity actions. Therefore, it is always assured that the user is able to interrupt the (possible) cycle by disabling the context-aware modality or navigating to another page.

4.4.2 Context-Aware Page Computations

To better clarify the interleavings and cross-effects between user navigation actions, automatically generated page requests, and adaptivity actions, this section shows some examples of adaptive page computations. The examples show that the page modeling logic needs to be well understood, when pages are context-aware, since (apparently) similar configurations of pages and context clouds can generate different, unexpected behaviors. The designer needs thus to carefully choose the most appropriate modeling solution.

Consider the page `Building` in Figure 4.11, and assume it not to be "context-aware", in the sense that it does not reflect any change of context. Although the figure has thus to be interpreted "without" the C-label, the page however reads context-specific meta-data, as represented by the `Get User` and `Get Area` units. The two units in fact retrieve the current user, and its current (logical) geographical area, e.g. inside a university campus, by navigating the `User2Area` relationship (we suppose that it is possible to associate a building of the campus to each area). The page can be accessed along two links. `Link1` does not carry parameters to the page, while `Link2` carries a parameter containing an

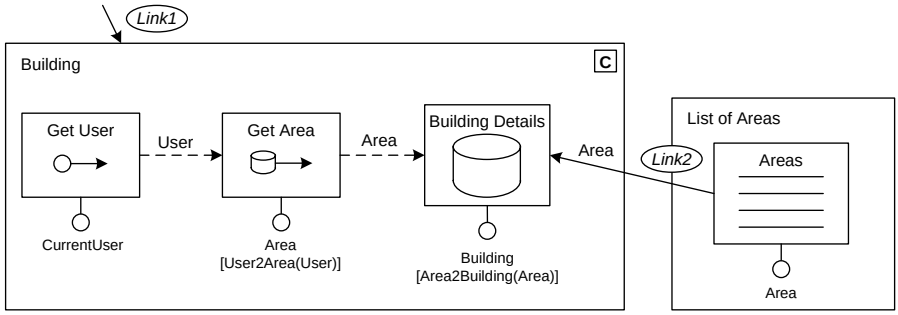


Figure 4.11: Hypertext schema presenting either an adaptive or a non-adaptive behavior at runtime. `Link1` and `Link2` represent possible user navigations toward the context-aware page.

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 the "standard" specificity rules described by items 1–3 in Subsection 4.4.1, is to show the details of the building of the user-selected area if the page is accessed along `Link2`; otherwise the idea is to show the building details associated to the user's current area. In particular, when `Link2` is traversed, the specificity of the user-selected area prevails in the computation of the `Building 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 like a context-aware page, as it is indeed represented by the C-label in the figure. The following behavior occurs (independently from the chosen adaptivity policy):

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

- *Page access along Link2.* 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 after 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,

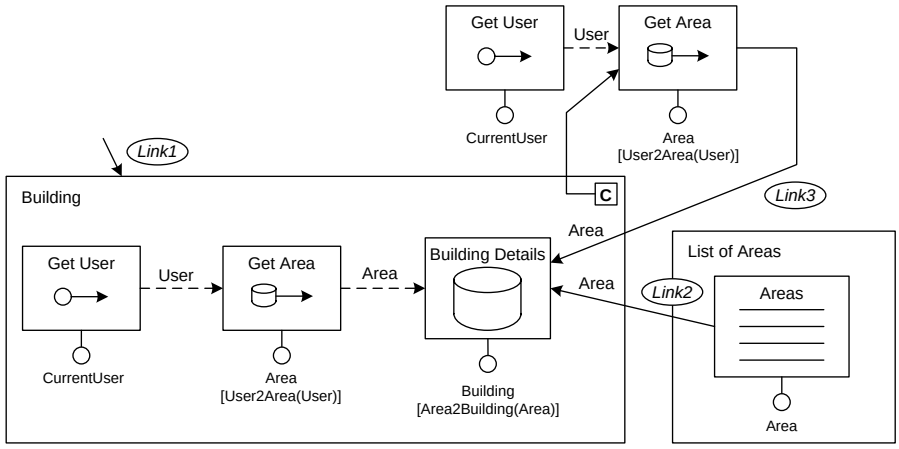


Figure 4.12: Overwriting out-of-date link parameters. `Link1` and `Link2` represent user-navigated accesses to page `Building`. `Link3` transports fresh context data, retrieved in the context cloud, which overwrite past user choices.

therefore, for 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 4.12. In this way, adaptivity actions are given a higher priority than past user selections. Assuming a *deferred* adaptivity policy, the page would be computed as follows:

- *Page access along Link1.* The page `Building` is accessed through a link which does not carry any parameters, and the page shows the details of the building 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 Area` unit inside the page.

- *Page access along Link2.* The page `Building` 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 Area` 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 `Building 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 Area` 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 in 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 in 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 modeling 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 described in Figure 4.11, and *dynamic with uniform refresh semantics* as modeled in Figure 4.12.

4.5 Discussion

In this chapter we extended the WebML method to support the modeling of context-aware and adaptive Web applications. The extended visual language allows designers to specify context-aware Web applications in complete accordance with the conventional WebML design style, since the introduction of the new features and primitives does not entail any new modeling formalism or paradigm. This is one of the main differences between the approach proposed in this dissertation and other conceptual modeling approaches, which prevalently base their adaptivity features on proper new rule languages (e.g. [44]). Also, existing WebML solutions, such as content and operation units, can be fully leveraged for the specification of adaptive behaviors and, therefore, seamlessly extend from non-adaptive to adaptive hypertext specifications.

WebML already supports the *adaptability* of WebML applications to different delivery channels (i.e. devices) and user preferences. The introduction of *adaptivity* into WebML on one hand enables designers to model features that could not be specified before and, on the other hand, also allows users to take advantage of WebML's adaptability capabilities during runtime (e.g. by means of the `Change SiteView` or `Change Style` units), thereby blurring the borders between *adaptability* and *adaptivity* as intended, for example, by Frasincar et al. [8].

It is worth noting that, although at a first glance it seems that the described extension was achieved mainly by developing new, context-specific units, the actual novelty is represented by the interpretation of context as autonomous actor over the application's hypertext front-end. This feature and the possibility to hierarchically structure adaptivity actions into context-aware areas and site views required a revision of the *computation logic* for context-aware pages, as discussed in Section 4.4. On top of this new execution framework it is now possible to easily define additional new context-specific units or operations addressing possible new requirements of individual applications or of specific application domains. The introduced modeling solution thus guarantees the *extensibility* and *customizability* that characterizes the WebML modeling approach in general.

5 Implementing Adaptivity and Context-Awareness

The research described in this dissertation has been conducted in the context of the Italian research project MAIS (Multichannel Adaptive Information Systems [68, 69]), during which also two prototypes were developed. Prototype development proceeded in two complementary steps: the first step produced a *proof-of-concept* demo application based on an external implementation of the adaptivity features (i.e. without altering the WebML runtime environment); the second step resulted into an extension of the *WebRatio CASE tool* [2].

In this chapter we briefly describe the two prototype implementations. We in particular concentrate on how the modeling concepts presented in Chapter 4 have been implemented as extension of the WebRatio CASE tool. Such an extension enables the tool-assisted specification of context-aware Web applications and, consequently, the automatic generation of the application code, starting from the extended WebML schemas. Therefore, the resulting model-driven approach almost entirely covers the whole development process – from data design to implementation.

In this chapter we also propose a possible further extension enabling context monitoring in the *background* (i.e. without using continuous refreshes for context monitoring), thus avoiding unnecessary page re-computations that could be perceived as intrusive by end users.

5.1 Pre-Processing of Page Requests

In order to show the implementability of the new concepts introduced at the conceptual level (i.e. at the data and hypertext level), we have developed a first prototype [70] on top of the existing WebML runtime environment that consists in a demonstration application. The prototype adopts an *external* solution for the implementation of the context-aware features described in the previous chapter, as it does not require any modification to the existing runtime environment of the WebRatio tool and instead is based on a pre-processing mechanism: page requests ar-

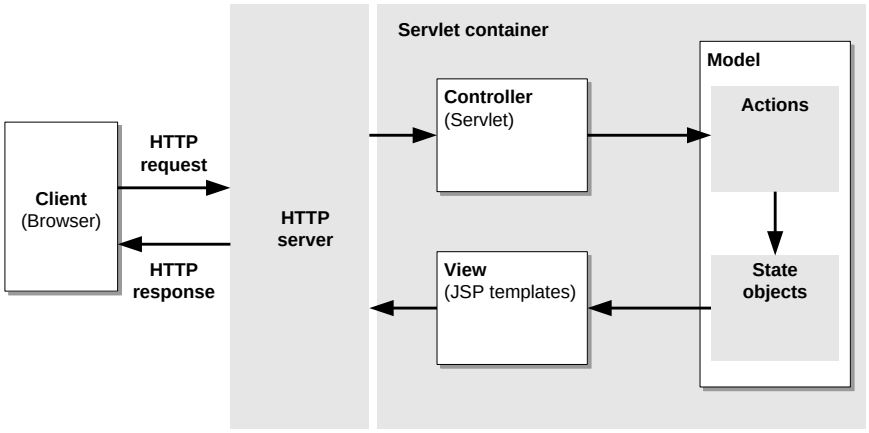


Figure 5.1: The MVC architecture applied to Web applications.

riving at the Web server are interpreted and – if necessary – adapted, before the actual request is passed to the WebML runtime environment for page computation. The runtime environment is therefore completely unaware of the adaptivity to be provided.

Figure 5.1 graphically depicts the MVC (*Model-View-Controller*) design pattern applied to the domain of the Web and adopted by the WebML runtime environment[1]: the *Model* contains the business logic of the application (i.e. the WebML unit and page logics), the *View* contains the presentation logic to create proper user interfaces (i.e. a set of JSP files) and the *Controller* manages the interactions triggered by the users' actions. According to this internal architecture of WebML applications, the pre-processing of page requests can be achieved by extending the Controller with adaptive logic.

The addition of adaptive logic (according to the context clouds in the extended WebML schemas) to the Controller enables the Controller to accept arbitrary HTTP requests and to check whether they refer to context-aware pages (as indicated by the presence of the `automatic` parameter – see Subsection 4.4.1) or to conventional ones. In case of requests for conventional pages, the Controller simply forwards the request to the Model for page computation; in case of context-aware pages, the Controller first performs the possibly associated adaptivity actions, modifies the request string to reflect possibly new computed page parameters (e.g. as required in the case of adaptive page contents) and, then, forwards the request to the Model for computation.

The external solution fully reflects the new page computation logic

[1] More specifically, WebML is implemented in the J2EE/Struts MVC framework [71]

outlined in Figure 4.10 and takes full advantage of the existing WebML runtime environment. Controller and adaptivity actions are hand-coded and allow the application to manage (i) the access to context data, (ii) the adaptation of page contents by overwriting request parameters, and (iii) the automatic navigation to other pages of the application by redirecting page requests. Although Controller logic and page-specific adaptation logic are intermixed and hardwired (i.e. they lack support for visual design and automatic code-generation), this first proof-of-concepts prototype has allowed us to assess the feasibility of the proposed conceptual modeling solution and to set up an experimentation environment for simulating and testing its viability. The prototype showed us that in principle the introduced ideas are applicable in practice, independently from the chosen conceptual modeling method (if any), and that the desired features can be implemented by means of standard Web technologies.

5.2 Implementing Context-Awareness in WebRatio

The second prototype [72] addresses the shortcomings of the first prototype and provides an extension of the WebRatio CASE tool to fully reflect the proposed visual design method. The implementation exploits WebRatio's native extension mechanism [73] that allows developers to add 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 [58] and the design of workflow-driven hypertexts [64].

In particular, the implementation of this second prototype has occurred along two complementary dimensions: the first dimension has concerned the introduction of context-aware pages as described in Subsection 4.3.1, the second dimension has referred to the *adaptivity actions* described in Subsection 4.3.6 and to be applied in the context cloud. Implementing context-awareness in WebRatio thus requires a good understanding of the software architecture of WebML applications and of WebRatio's code generation logic.

5.2.1 The Architecture of WebML/WebRatio Applications

All the applications generated with WebRatio share the same software architecture, which is a classical three-tier architecture taking advantage of the MVC design pattern; Figure 5.2 graphically depicts the internal architecture of automatically generated WebML applications. Each page

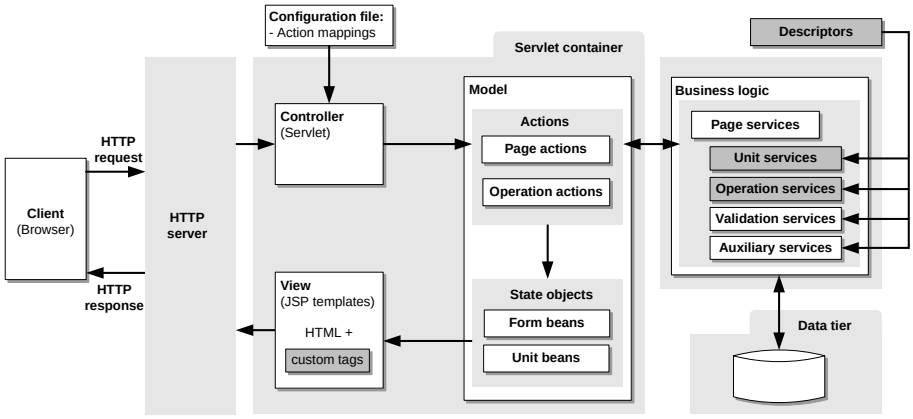


Figure 5.2: The runtime architecture of a WebML application automatically generated with WebRatio [73].

of an application consists of four elements: (i) a *page action* in the Model; (ii) a *page service* in the business tier; (iii) a *JSP template* in the View; and (iv) a *page action mapping* in the Controller's configuration file.

The *page action* is an instance of a Java class, which is invoked by the Controller when the user requests the page; the page action class extracts the input from the HTTP request and calls the page service in the business tier, passing to it the needed parameters. When the page processing terminates, the page action notifies the Controller that the page content is ready to be displayed.

The *page service* is a business function supporting the computation of the content of a page. It exposes a single function `computePage()`, invoked by the page action to carry out the parameter propagation and unit computation process. At the end of the page service execution, a set of Java objects storing the content of the page units (called unit beans) is available to the View.

The *page template* is a JSP template, which computes the HTML page to be sent to the user, based on the content of the Model. It contains the static HTML needed to define the layout where the units are positioned, and custom tags implementing the rendition of the content of units.

Finally, the *action mapping* is a declaration placed in the Controller's configuration file that ties together the user's request, the page action, and the page template.

Each WebML content unit maps into two components of the MVC architecture: a *unit service* in the business layer and a set of *custom tags* in the View. WebML operation units map into three components of the

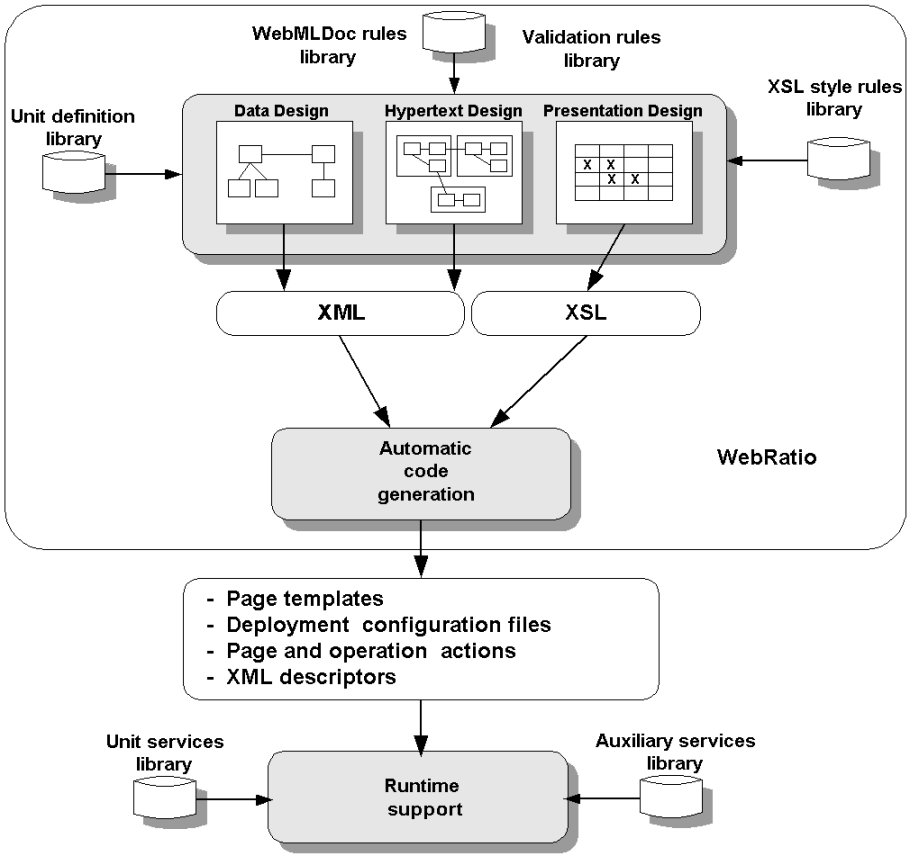


Figure 5.3: Overview of the WebRatio modeling environment and code generation process [73].

MVC architecture: an *operation action* in the Model, which is similar to a page action, an *operation service* in the business layer, which is similar to a content unit service, and an *action mapping* in the Controller's configuration file, which dictates the flow of control after the operation action completes the execution.

5.2.2 Extending the WebRatio CASE Tool

The extension of the WebRatio design environment, i.e. the development of custom units, takes the developer through all the components of WebRatio, because the definition and the execution of a unit requires addressing both design-time and run-time issues, as graphically summarized in Figure 5.3.

Adding a custom unit to the design environment may require:

- Adding a *unit definition* to the unit library (mandatory). This enables WebRatio to place the custom unit in the hypertext diagram, link it to other units, and define the coupling of input and output parameters.
- Adding a set of XSLT rules for *validating* the usage of the custom unit in the hypertext diagram and for producing error and warning reports (optional).
- Adding a set of XSLT rules for *documenting* the usage of the custom unit in the WebMLDoc project documentation (optional).

Adding new units to WebRatio also requires enabling the WebRatio code generator to handle custom units by:

- Adding a set of XSLT rules for producing the runtime *XML descriptors* associated with the custom unit (optional).
- If the custom unit is a content unit, adding one or more unit *presentation cores*[2], which are XSLT rules for producing the server-side tags or scripting instructions to be inserted in the page templates (mandatory).

Finally, deploying a custom unit into the WebRatio runtime environment requires implementing a *runtime Java class*, which actually performs the business service to be provided by the new unit. During runtime, the generated unit descriptors are used to instantiate the Java class for page computation.

5.2.3 Implementation

The new concepts introduced in Section 4.3 do not all boil down to the implementation of new custom units. For instance, the new page logic to be adopted when computing adaptive pages, or the possibility to add context clouds as well to areas and site views, not just to pages, would require extensions to the WebRatio environment that could only be achieved by significantly altering the source code of the tool. The implementation of the adaptivity ideas presented in this dissertation represents a compromise between the conceptual extension of the WebML language and the native extensibility of the WebRatio design environment. More precisely:

[2] Please refer to the EasyStyle User and Reference Guide for a in-depth explanation about the definition of new unit cores [74].

- The implementation of the new *page logic* and the tagging of context-aware pages with the C-label is implemented by means of a new content unit, the so-called `Context` unit; the unit is used in place of the C-label and enables the triggering of the adaptivity actions specified in the context cloud during the computation of the page. The unit also contains the parameter passing logic and manages the polling mechanism, granting the context cloud control when required and according to the chosen adaptivity policy.

- The `Get ClientParameter` unit is implemented as content unit to be placed inside a page, but without visual rendering in the page, because only content units have access to page parameters (operation units are executed in an isolated fashion).

  In the prototype implementation, the communication of client-side sensed data occurs by appending parameters to the request string of the page request (i.e. the *query* of the URL). Other communication mechanisms, e.g. based on the SOAP communication protocol, are under investigation.

- The `Get Data`, `Change Site View` and `Change Style` units are implemented as standard operation units.

The screenshot in Figure 5.4 refers to the extended visual WebRatio environment and shows a WebML model fragment of the application discussed in Chapter 6. The context-aware page `Buildings` contains the aforementioned `Context` unit (placed in the upper right corner of the page), which takes in input the client-side parameters `longitude` and `latitude`, accessed by means of the two `Get ClientParameter` units, and forwards them to the context cloud. The context cloud associated to the page consists of the three units `GetArea`, `GetBuild` and `GetRoad`. The link directed from the `GetBuild` unit to the `Building` unit inside the page represents an automatic adaptation of the page content, while the link exiting the `GetRoad` unit and directed to the unit inside the other page represents an automatic navigation action. The reader is referred to Chapter 6 for a more detailed description of the modeling example.

5.3 Enabling Background Context Monitoring

So far the concept of *active context-awareness* was based on the idea of periodically refreshing a viewed page, so as to grant the application

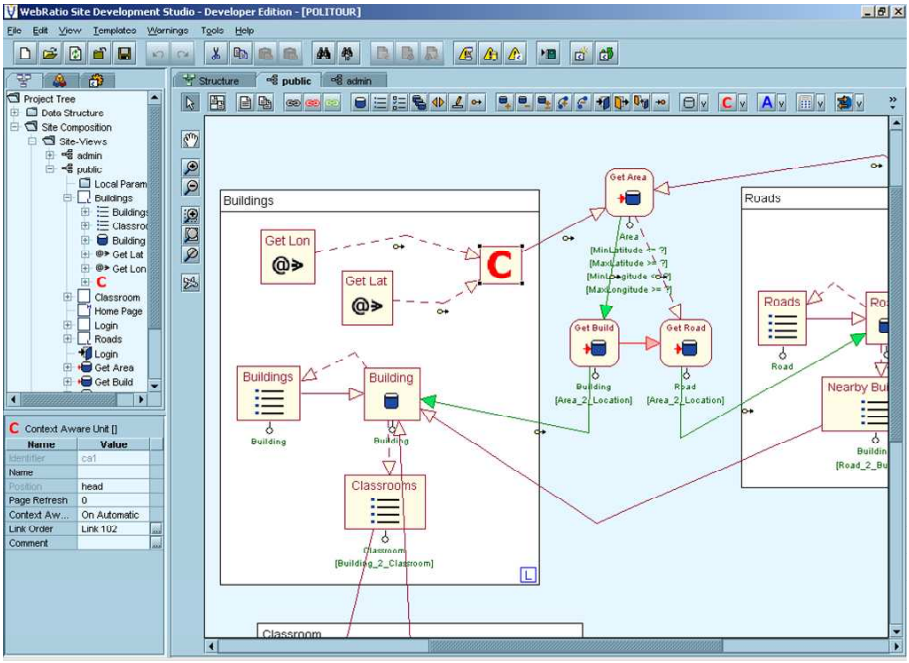


Figure 5.4: Screenshot of the extended visual environment of the WebML CASE tool.

the possibility to monitor context data[3] (i.e. check for changes and evaluate possible conditions). The periodic refresh of a page, however, also implies the new rendering of the page in the client browser, which in general is perceived as annoying by users viewing the page. To overcome this ergonomic shortcoming, in the following we describe a *background* context monitoring solution that has been developed as extension of the previous prototype implementations.

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

5.3.1 Context Monitor

Figure 5.5 shows a functional architecture for adaptive Web applications that extends the described architecture of WebML applications (see Fig-

[3] This solution is driven by the peculiarities of Web applications, where the application is "active" only during the computation of a page, as opposed to traditional desktop applications, which are in continuous execution.

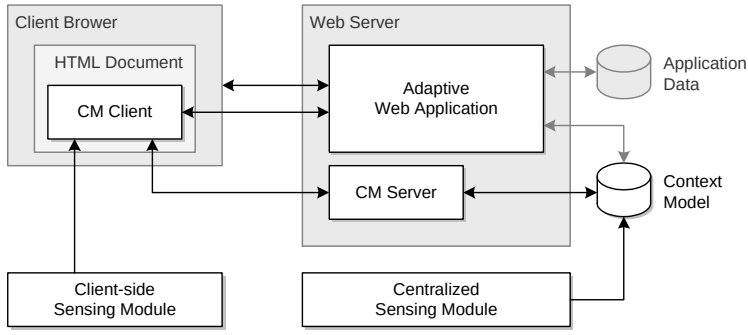


Figure 5.5: Functional architecture for background context monitoring.

ure 5.2) with a new client-server module, called *Context Monitor* (CM), providing the necessary context monitoring logic. As further depicted by the figure, in case of client-side context sensing, the CM module also enables the communication of client-side sensed context parameters, which could be required at the server side to evaluate context changes and/or conditions over context parameters.

The CM consists of two separate modules, one on the client side and one on the server side. The CM Client module is a piece of business logic embedded into the page's HTML code and executed at the client side (e.g. a JavaScript function, a Java applet, or a Flash object), while the CM Server module works in parallel to the Web application on the same Web server. The CM Client is in charge of periodically monitoring the context state and deciding whether possibly occurring context variations demand for the adaptation of the currently viewed page.

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

5.3.2 Page Context Parameters

In line with the idea of *Page context* (cf. Subsection 4.3.4), the CM focuses its attention only to the subset of context data in the context model that really determines the adaptive behavior of the viewed page. While the (abstract) definition of Page context primarily was intended to support the design phase of context-aware applications, the CM aims to support the context monitoring activity during runtime. This implies explicit knowledge about the pages' Page context, which can be achieved by defining proper Page context parameters for each context-aware page:

Definition 12 (Page Context Parameter) Page context parameters *define the view over the context model that captures all the static and dynamic properties of a page's* Page context *by means of suitable queries over the context model.*

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

5.3.3 Context Digest

In oder for the CM to be able to decide whether adaptivity is required, changes to the Page context (i.e. the Page context parameters) must be communicated from the CM Server to the CM Client.

In order to enhance the efficiency of the overall context monitoring activity, the state of the Page context is not communicated from the CM Server to the CM Client in form of the set of Page context parameters, but instead it suffices to transmit and compare a numeric digest computed over the respective Page context parameters, as each change to the values of the Page context parameters also results in a change of the numeric digest. We call such a numeric digest *Context digest*:

Definition 13 (Context Digest) *The* Context digest *corresponding to the Page context of a page is the numeric checksum computed over the ordered list of Page context parameters.*

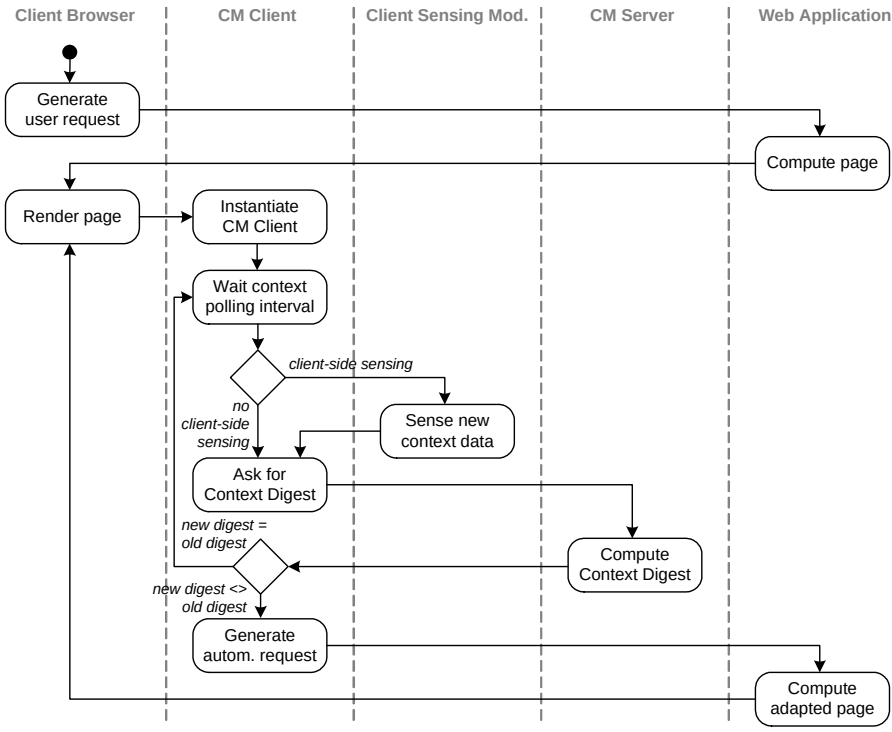


Figure 5.6: Background context monitoring for active context-awareness (with client-side context sensing): communicating context data and triggering adaptivity.

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

Note that, as already hinted at in Subsection 5.3.1, sending requests for fresh Context digests to the CM Server also enables the CM Client to transmit fresh client-side sensed context data, which can be used during the computation of the parametric Page context queries and, thus, for the computation of the Context digest.

Figure 5.6 details the resulting flow of activities enabling the active behavior of the application and shows how the single modules cooperate in order to determine whether adaptivity is required or not. The diagram has one start node (`Generate user request`), which corresponds to the user's navigation to a C-page, and no end node, since the cycle in the

lower part of the diagram is only interrupted by an explicit user navigation leading the user to another C-page (which corresponds to starting again from the start node of the diagram and to monitoring the Page context of the new page) or to a conventional page (which does not cause any context monitoring activity).

Note that the client-side sensing module and the sending of fresh context data only apply to applications that sense context data at the client side; if only a centralized or server-side sensing mechanism is adopted, the step `Sense new context data` and the possible communication of fresh context data are omitted. For example, a centralized, RFID-based sensing infrastructure directly updating the context model does not require any transportation of context parameters from the client to the server.

The described mechanism assumes that connectivity is available during the viewing of a C-page in order for the CM client to be able to communicate with the CM server. In case of intermittent connectivity, which is a very frequent situation in mobile environments, the CM client keeps working by periodically polling the CM Server, despite the absence of connectivity. The CM Client is however programmed to manage possible lacks of connectivity and therefore does not generate errors, with the only side effect that adaptivity is suspended until the connectivity is restored.

5.3.4 Context Monitor Implementation

The CM is implemented as a client-server module, completely independent from the implementation of the Web application. Despite its integration into the WebML runtime environment as described in this work, its function is general in nature and, given access to the context model, only demands for explicitly invokable page adaptation operations (e.g. via HTTP).

In our current implementation, the CM Client is a Macromedia Flash object[4]. Its configuration is performed directly within the HTML code sent to the client browser and mainly consists in the specification of the context parameters to be sensed at the client side, as well as a suitable polling interval.

The CM Server, on the other hand, is implemented as a Java servlet

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

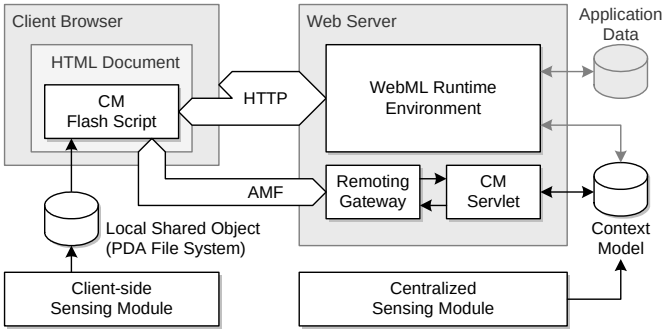


Figure 5.7: Context Monitor Implementation.

communicating with the CM Client via the *Flash Remoting Gateway* [5]. The CM Server generates the digests representing the fingerprint of the current Page context state corresponding to the currently viewed page. For each context-aware page, an XML configuration file contains:

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

2. For each Page Context parameter, the query for extracting the specified context parameter values from the context data. If client-side or server-side parameters are available, they may be used for formulating selection conditions.

The configuration of the CM Server module thus consists in the declaration of the Page Context parameters that are used to compute the context digest and in the specification of the respective DB queries that associate a value to each parameter. At the current state of the work, this configuration still needs to be hand-coded by designers, but we envision a visually assisted specification (e.g. by means of a proper wizard) of Page Context parameter queries. A similar approach is for example already supported by WebRatio for the specification of derived relationships between data entities.

Figure 5.7 graphically depicts the described implementation of the Context monitor. The adaptive and non-adaptive hypertext pages, as well as the adaptivity actions specified in the previous section, are hosted and executed by the *WebML runtime environment* [2, 71] described in

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

Subsection 5.2.1. Chapter 6 provides some more details about the configuration of the Context monitor by discussing a case study.

5.4 Discussion

Potentially, each conceptual application model (i.e. data and hypertext schemas) could be hand-coded by programmers on top of different Web architectures and by using different technologies and programming languages, but the existence of a powerful CASE tool, such as the WebRatio tool suite [2], significantly facilitates the automatic code generation of WebML applications for the J2EE/Struts framework. The two described prototype implementations developed in the context of the MAIS project demonstrate the consistent implementability of adaptive hypertext schemas and the suitability of the newly introduced modeling constructs also for automatic code generation.

Due to some restrictions of the current implementation of the runtime environment, it was not yet possible to implement context-aware containers, i.e. areas or site views. However, in the current extension of WebRatio typical area-level adaptivity actions (e.g. accessing volatile context data and/or storing persistent context data) can be specified as page-level adaptivity actions. Although this solution may augment the redundancy of adaptive hypertext schemas, it allows functionalities that typically would be provided by context-aware containers to be specified at page level. Support for context-aware containers is planned for future prototype versions.

With respect to the first proof-of-concept prototype, the second prototype implementation has allowed us:

- To support the design of arbitrary context-aware Web applications and the automatic generation of the respective program code.
- To almost entirely reflect the adaptive design method proposed in Chapter 4 (context-aware containers are not supported).
- To capitalize on the WebML CASE tool and runtime environment.

In addition to the automatic code generation of adaptive hypertext schemas, also the background context monitoring solution implemented in the Context monitor module can be automatically included into the JSP page templates of adaptive pages and (partially) configured during the code generation phase. This allows designers to augment the usability of adaptive Web applications with little additional development effort.

Since code generation starts from WebML schemas, only WebML-related concepts can be automatically coded. The automatic generation of the context sensing logic or the supply of proper universal software modules, which would allow the extended CASE tool to cover the whole development process of context-aware applications, is not feasible, due to the high heterogeneity of both hardware and software aspects of the sensor devices as well as to the high dependency from the individual application domain. To assist application designers in this task, however, three easy to use interfaces for communicating sensed context data to the application have been defined: (i) direct updates of the context model by means of conventional *SQL statements*, (ii) a simple *shared file* to exchange client-side context parameters between the client-side sensing software and the CM Client module, and (iii) HTTP *session parameters* for server-side context parameters. Using these interfaces allows the designer of the sensing infrastructure to disregard how the application manages context parameters and to focus on the development of the sensing software.

6 Case Study

The MAIS project, in the context of which the ideas elaborated in this dissertation have been developed, was officially concluded in June 2006 with a publicly accessible demo and presentation day held at the Politecnico di Milano, Italy. To demonstrate the conceptual extension of WebML to model context-aware Web applications and the extension of the WebRatio CASE tool, we implemented a proper context-aware demo application (called *PoliTour*), supplying location-aware information about the roads, buildings, and classrooms in the Politecnico university campus. The event was held in one of the arcades of the Politecnico campus, which for the event had been covered with a WiFi connection. By means of this WiFi connection, the PoliTour application could be accessed through a PDA device equipped with a GPS receiver for (outdoor) location sensing. During the workshop day, the demo was complemented with a poster of the overall methodology for the development of context-aware Web applications.

In this chapter we build on this demo application to exemplify the concepts introduced in the previous chapters and to highlight how the design of the context model impacts also on the design of the adaptive hypertext.

6.1 Conceptual Design

The PoliTour application leverages two different pieces of context information to support the context-aware delivery of application contents: (outdoor) user *position* and WiFi *connection quality*. Sensed context data are thus the geographical longitude and latitude for user positioning and the signal strength of the available WiFi connection; both position and signal strength are sensed at the client side. Position data is used to provide users with location-aware data, i.e., as the user moves around the campus, the application publishes location-aware details about nearby buildings and roads. The connection quality is used to alert users of low connectivity conditions, i.e., when the user is about to leave the WiFi-covered area.

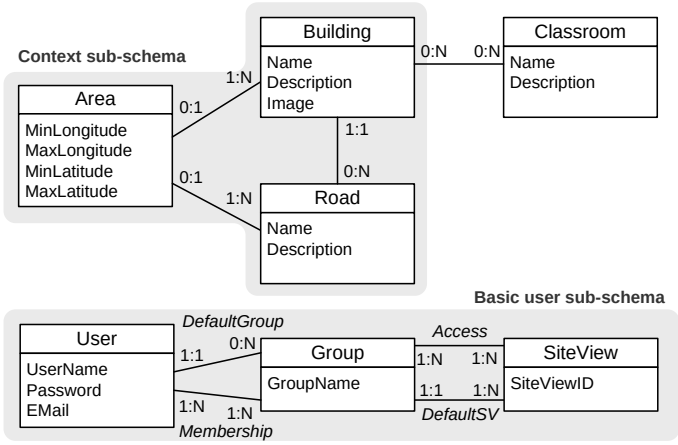


Figure 6.1: PoliTour data model when using volatile context parameters.

The former adaptivity implies the automatic adaptation of visualized contents and automatic navigation actions, while the latter is achieved by appropriately changing the CSS style sheet associated to the application (in our case, by changing the application's background color).

In the following, we first describe how these requirements can be modeled as data, then we show how the adaptive hypertext schemas can be modeled on top of these data.

6.1.1 Data Modeling

As discussed in Section 4.2, context data can be modeled in different flavors, where one of the most interesting decisions to be taken – also impacting hypertext design – is whether individual context parameters are to be modeled as *volatile* or as *persistent* context data. While the former do not require any explicit definition in the context model, the latter do require a proper definition as context data.

To show the implications of these alternative design choices, in this case study we discuss both, and treat position and connectivity data once as volatile data, once as persistent data. More precisely, the decision impacts on the design and use of the client-side sensed context parameters `longitude`, `latitude` and `RSSI` (Received Signal Strength Indicator).

Volatile Parameters

Figure 6.1 illustrates the data schema of the PoliTour application in case of volatile context parameters to be used in the hypertext model.

The entities `User`, `Group`, and `Site View` represent the basic user model for WebML applications. Since the sensed physical context parameters are volatile, the context sub-schema in the figure only contains logical context data.

In order to translate the physical longitude and latitude into meaningful location information that can be used to determine the adaptive behavior of the application, we divide the campus area into a set of contiguous, rectangular areas. We then map roads and buildings onto these areas. Reading a user's current longitude and latitude allows then the application to associate an area to the user's position, which, on the other hand, allows the application to identify the closest building or road. The almost exact west-east orientation of the Politecnico campus eases the definition of the elementary areas: four attributes characterize the entity `Area`: min and max longitude, and min and max latitude.

The entities `Road` and `Building` provide a further level of logical context data. Navigating from the `Area` entity to these two entities enables the application to derive the actual building or road to be visualized.

The entity `Classroom` is not part of the context model, as the application does not react to position changes at that level of granularity. It rather represents core application data. The entities `Road` and `Building` can as well be interpreted as application data, as they contain the actual core contents of the application, underlining the difficulty that may arise when one tries to strictly separate context data from (pure) application data.

Persistent Parameters

Figure 6.2 instead shows how the context model could be modified to capture the case of persistent context properties to be used in the hypertext model. The only difference with respect to Figure 6.1 is the addition of the two entities `Connectivity` and `Position`.

The entity `Position` contains for each user his/her personal longitude and latitude. The entity directly stores the physically sensed values. Users are not associated with areas, buildings, or roads at the data level; this association can be computed at runtime by querying the context model.

The entity `Connectivity`, on the other hand, also incorporates a translation of the physically sensed (continuous) RSSI value into a discrete WiFi connection quality, expressed by qualitative levels (i.e. "Low" and "High"). The entity is constituted by the three attributes `Level`, `minRSSI`, and `maxRSSI`, which allow the application to associate at runtime a suitable level to each user.

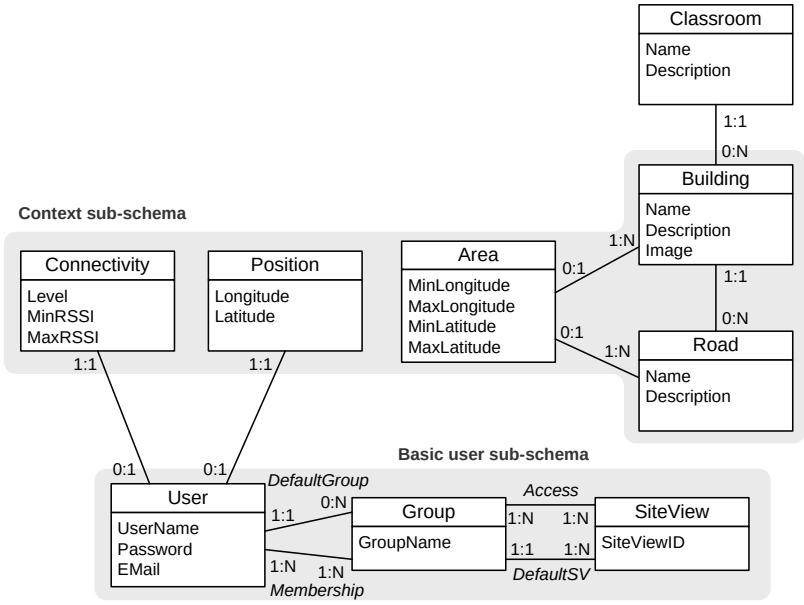


Figure 6.2: PoliTour data model when leveraging persistent context data.

6.1.2 Hypertext Modeling

While the difference between volatile and persistent context data is easily expressed by including or excluding proper entities in respectively from the context model, the two modalities sensibly differ when it comes to hypertext design. In the following we thus propose two different hypertext schemas, one for each approach, and show how the two modalities influence hypertext design. Before that, however, we describe the basic hypertext structure of the PoliTour application, so as to ease the comprehension of the subsequent discussion of the adaptivity features.

Figure 6.3 shows a conventional (i.e. non-adaptive) WebML hypertext schema, consisting of one site view and three pages, two of which are landmark pages, always accessible through the application's menu. In the home page `Buildings` the user may select a building from the `BuildingsIndex` unit, which causes the application to display the details of the selected building in the `BuildingData` unit and to provide the list of classrooms of that building in the `ClassroomsIndex` unit. If the user selects one of the classrooms, he/she navigates to the `Classroom` page, which shows the respective details. If instead the user accesses the `Roads` page through the menu, he/she is provided with the list of roads. Again, if he/she selects a road, the page displays the details and also provides the list of buildings associated to that road. Selecting a building

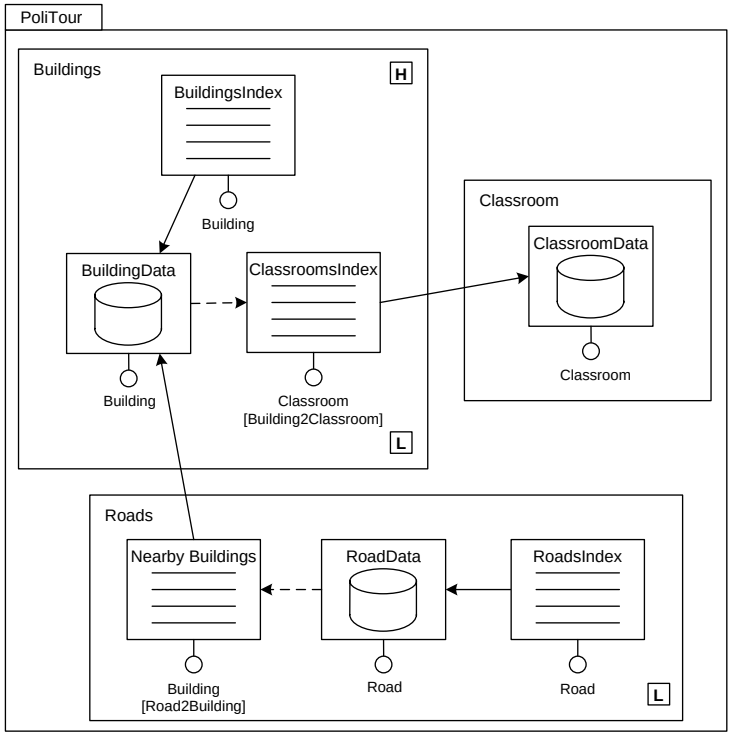


Figure 6.3: The non-adaptive hypertext model of the PoliTour application.

from the list, leads the user to the `Buildings` page.

The hypertext in Figure 6.3 does not present any adaptive behavior, as there are neither C-pages nor context clouds. Note that, consequently, this hypertext schema can be defined on top of each of the previous data models. Context-aware behaviors are introduced in the following.

Volatile Parameters

The hypertext schema in Figure 6.4 extends the one presented in Figure 6.3 and shows a possible solution to the modeling of the adaptivity requirements of the PoliTour application. Again, we have one site view and three pages, two of which are now tagged as context-aware pages, i.e. page `Buildings` and page `Roads`. The site view as well is context-aware and gathers adaptivity actions that are common to all contained context-aware pages.

The pages `Buildings` and `Roads` share the same adaptivity actions (i.e. the same context cloud) providing location-awareness to the displayed

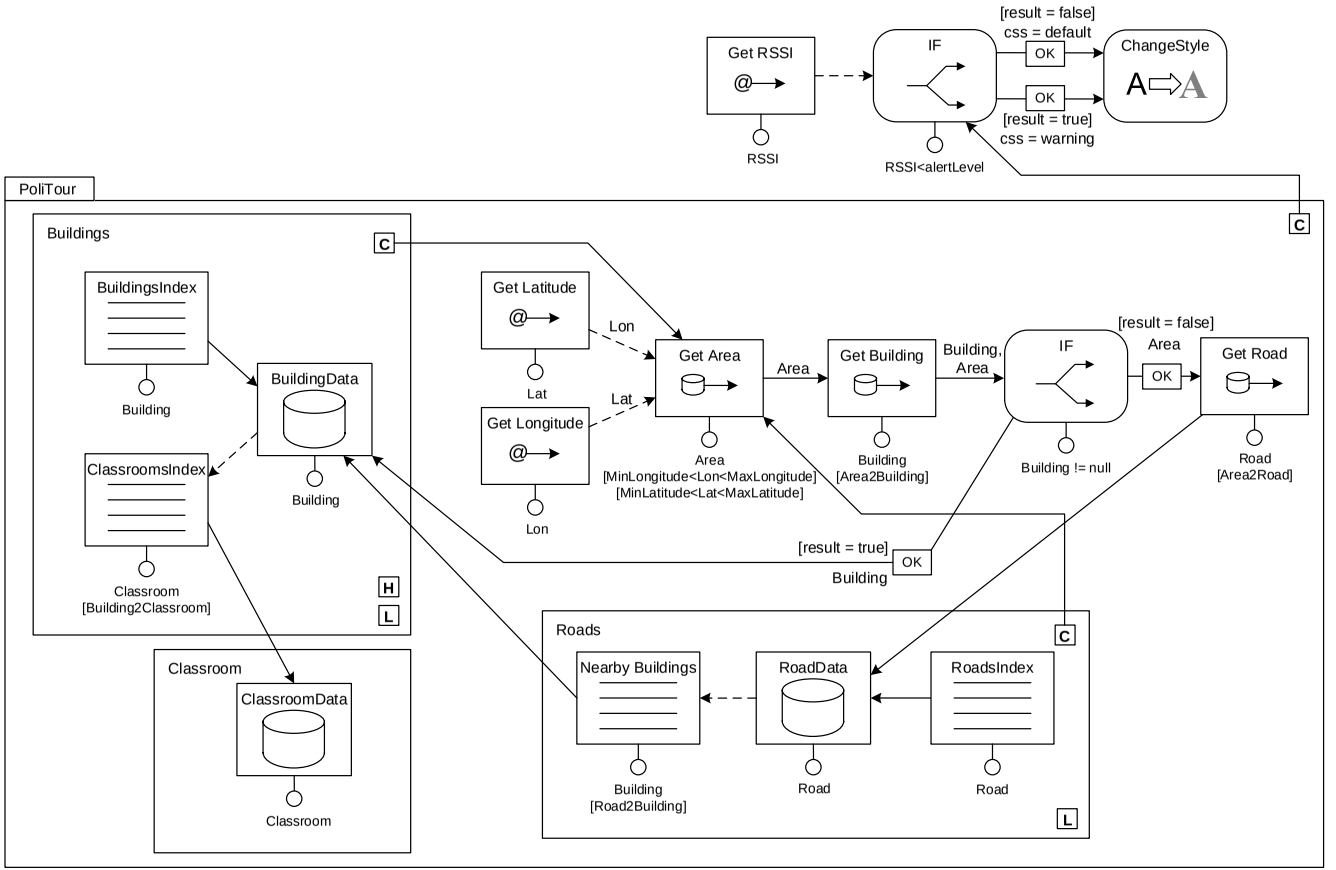


Figure 6.4: PoliTour hypertext model based on volatile physical context parameters.

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

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

The adaptivity actions associated to the surrounding site view specify the desired alert to be sent to users who are about to leave the WiFi-covered area. The `Get RSSI` unit accesses the volatile RSSI parameter sensed at the client side, and the `If` unit compares the retrieved value with a predefined level (`alertLevel`), below which the connectivity is considered low. In case of low connectivity, the style sheet `warning` is adopted, otherwise the `default` style sheet is adopted. For sake of simplicity, we therefore model the alert of low connectivity conditions by means of a `Change Style` unit. Under low connectivity conditions the application is rendered with a red background, under normal conditions the application is rendered with a gray background.

We recall that actions associated to containers are evaluated/executed before any action at the page level is started. Hence, in Figure 6.4 the actions associated to the site view are executed before the actions associated to the pages `Buildings` and `Roads`.

Persistent Parameters

Figure 6.5 shows an extension of Figure 6.3, which is based on the use of persistent context data. Although in the case of the PoliTour application persistent context data do not provide any additional benefit to users, the application scenario allows us to describe how the hypertext model must change in order to support persistent context data. As the careful reader will have noticed, in Figure 6.5 we added a new container: at

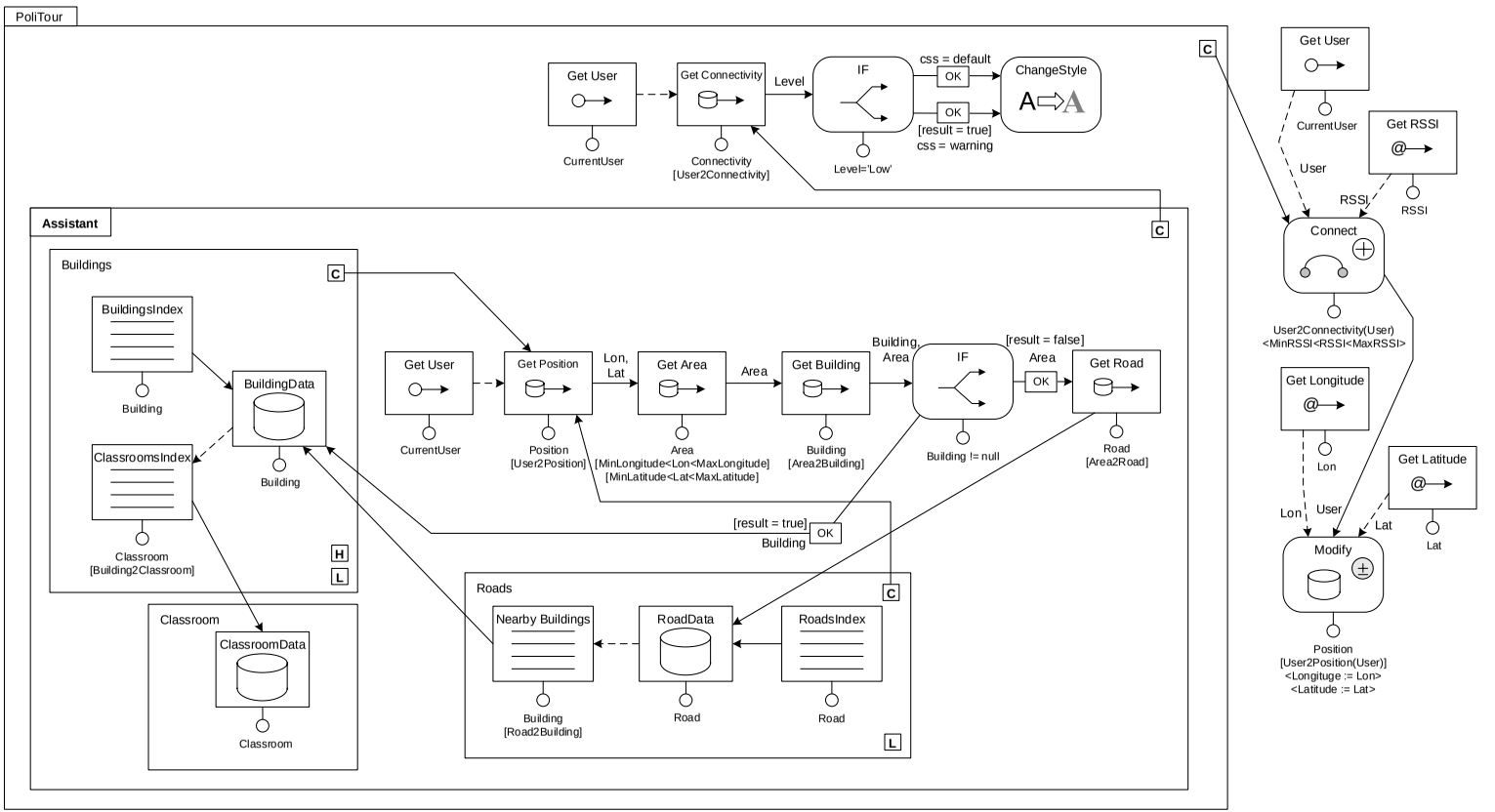


Figure 6.5: PoliTour hypertext model based on persistent context parameters.

the site view level we now store the volatile client-side parameters in the context model in order to transform them into persistent data, while the previous site view has been substituted with an area (area `Assistant`). Adaptivity is thus specified by means of a three-layered hierarchy.

More precisely, the adaptivity actions associated to the site view implement the necessary logic to update the personal context data of each user in the context model. Therefore, the operation chain starts with a `Get` unit that accesses the current user's identifier and a `Get ClientParamter` unit that accesses the RSSI parameter. The retrieved values are used to connect the user to a suitable logical connectivity level modeled by the `Connectivity` entity in Figure 6.2. We suppose that two different levels of connectivity are defined, i.e. "Low" and "High", although the context model would also allow the definition of finer granularity levels.

From the `Connect` operation we pass the `User` identifier to the `Modify` operation, which also takes in input the longitude and latitude values retrieved by means of the two `Get ClientParameter` units. The operation modifies the stored position data in the `Position` entity of the extended context model in Figure 6.2. From this point on, the specification of the actual adaptivity actions to be associated to the inner pages and the inner area is based on the use of the data stored in the context model and no longer on the use of volatile context parameters.

The two chains of adaptivity actions associated to the `Buildings` and `Roads` pages and to the context-aware area are analogous to the ones already seen in Figure 6.4, with two small differences: first, the `Get RSSI` unit used to access the RSSI value in the context cloud associated to the area is substituted with a `Get` unit to access the current user's identifier and a `Get Data` unit to navigate the context model and retrieve the logical connectivity level associated to the user. Second, the two `Get ClientParameter` units used to access the longitude and the latitude in the context cloud inside the area are substituted with a `Get` unit to access the user's identifier and a `Get Data` unit to extract the longitude and latitude parameters from the context model (i.e. from the entity `Position`).

Comparing the hypertext in Figure 6.4 with the one in Figure 6.5 allows one to identify some differences between the use of volatile parameters and the use of persistent parameters. Typically, the solution adapting volatile parameters yields less intricate hypertext schemas, while the solution with persistent parameters – according to our modeling guidelines – requires one more hierarchy level, i.e. the outermost level in charge of storing fresh context data into the context model. In the specification of the adaptivity actions, one can further see how in the case

of only persistent context data, access to individual (user) context data is achieved by navigating the data source from the entity `User` to the respective context entity. In the hypertext schema this is represented by the `Get` units that access the `CurrentUser` session parameter.

An Example Usage Story

Figure 6.6 exemplifies a possible interaction with the PoliTour application; both aforementioned hypertexts support the same application features and can be considered equivalent from the interface/interaction point of view.

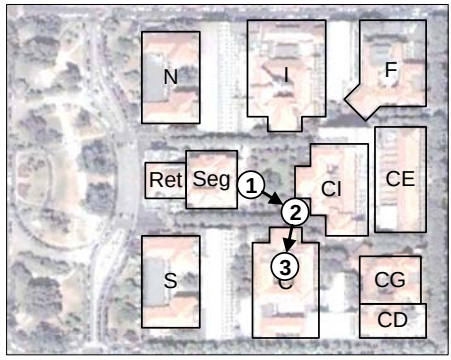
We assume the Politecnico campus is organized as represented in Figure 6.6(a), and that the user wants to move from location 1 to location 3, as highlighted on the map. Figure 6.6(b) shows the respective screenshots. The user starts from the central garden in the campus, moves to a nearby road and, finally, enters building *C*. The application automatically adapts the published contents accordingly. Once in the building, the user selects one of the classrooms of the building (see screenshot 4), thus he/she accesses the non-adaptive page `Classroom`. Turning back to one of the two context-aware pages `Buildings` or `Roads` would again enable the automatic adaptation of contents. Note that the only navigation action performed by the user in the described interaction is the selection of the classroom, while the adaptations of the page contents and the automatic navigation actions are context-triggered.

6.2 Implementation and Deployment

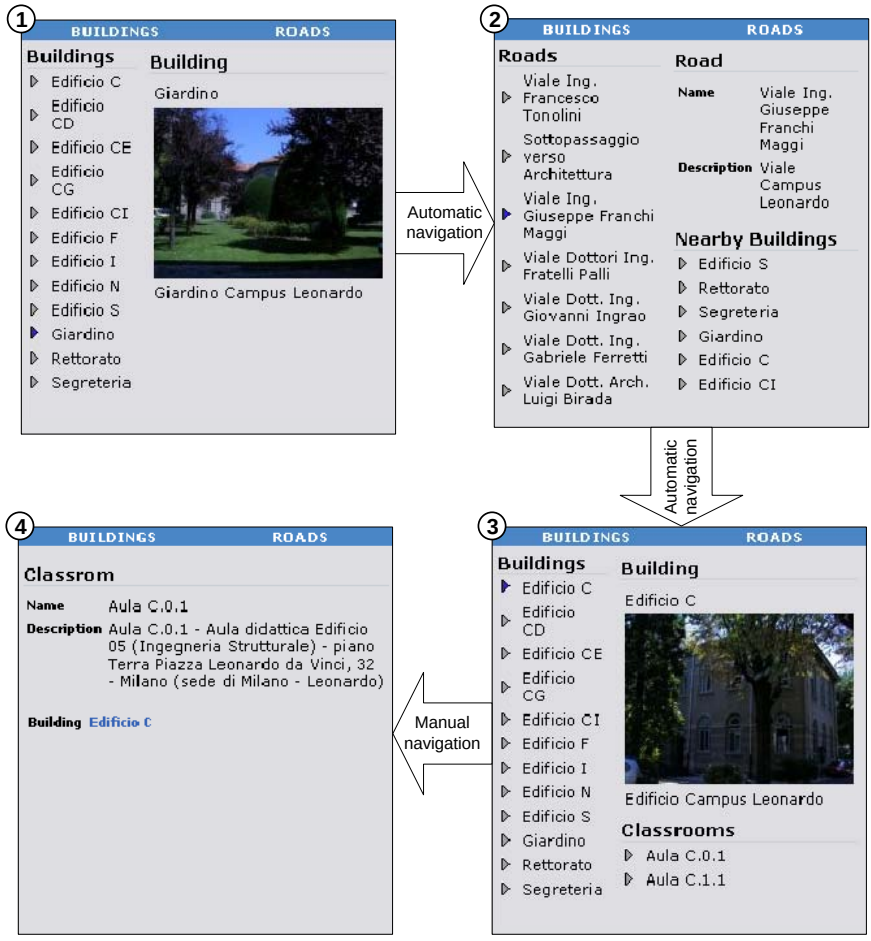
For the MAIS demo day, a hypertext solution similar to the one described in Figure 6.4 was adopted and implemented. The underlying data and context models were an intermediate version of the two models described in Subsection 6.1.1, where the entity `Connectivity` was explicitly modeled to highlight the translation of physical context data into logical context data. In addition, the background monitoring mechanism described in Section 5.3 was used to prevent unnecessary page refreshes. This solution further required the definition of suitable Page context parameters for the two pages `Buildings` and `Roads`.

6.2.1 Background Context Monitoring

The configuration of the Context Monitor module for the PoliTour application was performed as described in the following.



(a) Main campus of Politecnico di Milano.



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

Figure 6.6: The running PoliTour application.

Configuration of the CM Client

The CM Client configuration was achieved by means of the following (HTML) code lines added to the rendering of C-pages:

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

As described in Subsection 5.3.4, due to the client-side sensing, client configuration requires to specify the context parameters (`contextParams`) to be sensed at the client side, i.e. `longitude`, `latitude`, and `RSSI`, and the polling interval (`refresh`), i.e. 5 seconds. These parameters are automatically configured at design time through the WebRatio visual environment. The CM Client configuration requires also other parameters in order to work properly: `gatewayURL` represents the URL of the Flash Remoting Gateway, `userOID` represents the user identifier, and `currentURL` represents the currently requested page. These parameters are automatically configured at runtime by the execution environment.

Configuration of the CM Server

The following XML code presents the CM Server configuration file for page `Buildings` and page `Roads`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ConfigElements>

 <CAPage id="Buildings">
  <PageContextParams number="2">
   <param>
    <name>buildingOID</name>
    <query>
     select BUILDING.OID
     from   BUILDING, AREA
     where  AREA.toBUILD=BUILDING.OID
     and    AREA.MINLONGITUDE<?longitude?
     and    AREA.MAXLONGITUDE>?longitude?
```

```
      and    AREA.MINLATITUDE<?latitude?
      and    AREA.MAXLATITUDE>?latitude?
     </query>
    </param>
    <param>
     <name>connLevel</name>
     <query>
      select CONNECTIVITY.LEVEL
      from   CONNECTIVITY
      where  CONNECTIVITY.MAXRSSI>?RSSI?
      and    CONNECTIVITY.MINRSSI<?RSSI?
     </query>
    </param>
   </PageContextParams>
  </CAPage>

  <CAPage id="Roads">
   <PageContextParams number="2">
    <param>
     <name>roadOID</name>
     <query>
      select ROAD.OID
      from   ROAD, AREA
      where  AREA.toROAD=ROAD.OID
      and    AREA.MINLONGITUDE<?longitude?
      and    AREA.MAXLONGITUDE>?longitude?
      and    AREA.MINLATITUDE<?latitude?
      and    AREA.MAXLATITUDE>?latitude?
     </query>
    </param>
    <param>
     <name>connLevel</name>
     <query>
      select CONNECTIVITY.LEVEL
      from   CONNECTIVITY
      where  CONNECTIVITY.MAXRSSI>?RSSI?
      and    CONNECTIVITY.MINRSSI<?RSSI?
     </query>
    </param>
   </PageContextParams>
  </CAPage>
 </ConfigElements>
```

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

- The context digest of page `Buildings` is computed over the `buildingOID` and `connLevel` values, while the context digest of page

`Roads` is computer over the Page context parameters `roadOID` and `connLevel`.

- The value of `buildingOID` and the one of `roadOID` are extracted through parametric queries that use the `longitude` and `latitude` values provided by the CM Client[1] and retrieve the building respectively road associated to the current user location. In case the queries do not return any result, the value associated to the Page context parameters is *null*.
- The value of `connLevel` is extracted using the `RSSI` value provided by the CM Client. In case the query does not return any result, the value associated to `connLevel` is *null*.

With this configuration, the CM module is fully configured to work with the hypertext model described in Figure 6.4.

6.2.2 Automatic Code Generation

The (adaptive) hypertext and the CM Client configuration was automatically generated with the extended WebML code generator described in Section 5.2 and deployed on top of a J2EE platform. The configuration of the CM Server and the interaction with the sensing devices was coded manually. Its use is possible through PDA devices with wireless Internet access, using *Pocket Internet Explorer* with *Flash plugin* (for the CM module). The communication between the GPS module and the CM Client is implemented using the *Chaeron GPS Library* [75]; the WiFi received signal strength indicator (RSSI) is acquired on the PDA using *Place Lab* [76]. A demo of the PoliTour application is available at `http://dblambs.elet.polimi.it/politour/`.

6.3 Discussion

This case study allowed us to show how the modeling of context-aware Web applications can be performed in WebML and how design decisions regarding the context model impact on the design of the adaptive hypertext. The two discussed modeling examples represent two different levels of persistence of context data deriving from the use of either only volatile context parameters or only persistent context parameters. There are, of course, also intermediate modeling solutions that make use

[1] Variables expressed as `?name?` refer to either client-side sensed context data provided in input by the CM Client at each request for a new context digest, or to server-side session parameters available in the runtime environment.

of both volatile and persistent context data in the specification of the adaptive behavior of the application (e.g. persistent data only for history purposes and volatile data for feeding adaptivity actions), but these are not further elaborated in this dissertation, as – once the two described examples have been understood – they result to be quite intuitive.

Instead of always accessing client-side context parameters by means of the `Get ClientParameter` unit, it would also be possible to pass the respective values from the outermost context cloud to the inner ones by means of server-side parameters that can be read with conventional `Get` units. This however always requires the hierarchical specification of adaptivity and rather leads to a duplication of (volatile) context data. Passing parameters from one cloud to another is particularly powerful in those cases where the outermost cloud *computes* values that are reused in the inner cloud.

As for the configuration of the Context Monitor module, introduced to enhance the usability of context-aware hypertexts, this case study confirms its general nature. Independently from the particular hypertext model adopted, the Context Monitor only requires the definition of the Page context for each of the context-aware pages of the application and the definition of the client-side sensed context parameters to be sent from the client to the server, all properties of general validity in the design of context-aware Web applications, not deriving from the use of WebML as conceptual modeling language.

However, regardless of the level of persistence chosen for context parameters and, thus, regardless of the hypertext model that suits the chosen persistence level, the user remains completely unaware of whether volatile or persistent context parameters are adopted (provided that the possible hypertext models implement the same context-aware functionality). This observation underlines that the nature of the context model is hidden to the user, and that the user only perceives the effects of possible adaptations.

7 Exploitation and Evolution of Results

Throughout the entire definition process of the new features and primitives, the realization of context-awareness and adaptivity in WebML aimed at the development of a set of new concepts to be introduced in the language. The final goal was the extension of WebML to allow the modeling of context-aware Web applications in a fashion as aligned as possible with the preexisting modeling method. Context modeling and adaptivity modeling thus needed to be expressed in WebML terms, possibly requiring only a minimal set of new concepts.

Also, throughout this research, *extensibility* was always a concern during the extension of the modeling language, as extensibility facilitates the exploitation and the evolution of the proposed approach, also in light of parallel research topics that were tightly connected to the research described so far.

In this chapter we describe three different research areas in which the results stemming from the research described in this dissertation have been applied successfully: In the first place, the described research was performed in the context of the Italian research project MAIS [68], which was a joint research effort by several different research units from Italian universities and companies that collaborated in the development of a framework for multichannel, adaptive information systems. In this regard, the solutions described so far have been adapted to support the integration with other research groups. Next, we discuss an extension that we have built on top of the approach described in the previous chapters in order to support the design of behavior-aware Web applications. Last, we propose an evolution of the conceptual modeling approach for adaptivity and context-awareness, in order to overcome an intrinsic limitation: its pure design time applicability. Therefore, we outline our work on the development of a detached rule engine for runtime adaptivity management through event-condition-action rules. The first two sections thus prove the scientific validity of the outlined ideas, while the last section opens up new room for additional improvements and new research challenges.

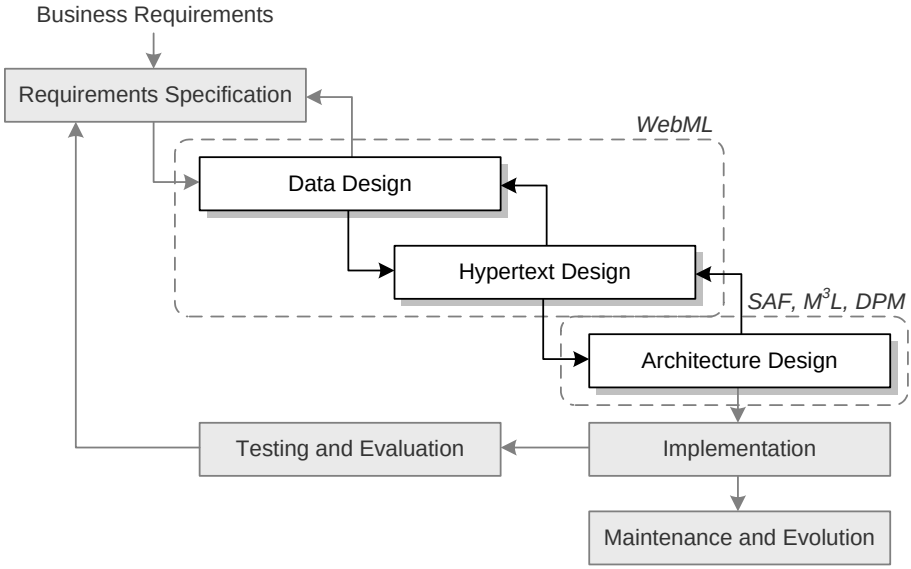


Figure 7.1: Design phases of multichannel or multimodal, adaptive Web applications. The labels of the dashed boxes highlight the design instruments adopted in the MAIS project.

7.1 Multichannel and/or Multimodal Adaptive Information Systems

Figure 7.1 shows how our results were integrated with the results from other two MAIS research teams[1], in order to provide application designers with both methodological and technological instruments for the design of multichannel and/or multimodal adaptive Web applications.

More precisely, the figure shows that *WebML*, that is, the extension introduced in the previous chapters, is adopted for the conceptual data design and hypertext design. Architecture Design (and Implementation), on the other hand, does not solely rely on the WebRatio tool and its code generator, but instead also leverages three different frameworks (i.e. SAF, M^{3}L, and DPM) developed by the project partners: *SAF* (Situation-Aware Framework) leverages the context-aware extension of WebML to provide context-aware features at the presentation level; the *DPM* (Dynamic Presentation Manager) supports the automatic adaptation of contents to different delivery channels; and *M^{3}L* (MultiModal Markup Language) supports the delivery of multimodal contents.

[1] Our group tightly worked together with *Cefriel* (Consorzio per la formazione e la ricerca in Ingegneria dell'Informazione), a Milan-based IT research consortium, and *Engineering*, a Palermo- and Rome-based IT company.

7.1.1 Adaptivity for the Presentation Layer

Cefriel developed the *Situation Aware Framework* (SAF [77, 68]) for the design, the delivery, and the execution of context-aware Web applications by means of adaptations in the presentation layer of the Web application. SAF is a complementary step in the design of adaptive Web applications with respect to the extensions of WebML described in this dissertation.

The kinds of adaptation supported by SAF are:

- *Layout*: adaptation of the arrangement of objects in the page space.
- *Presentation*: adaptation of color schema, font type, and font size.
- *Entity instance selection*: selection of a specific instance of an entity.
- *Attribute selection*: selection of the attributes to be shown by an entity.

In SAF, context-awareness is managed through a declarative approach in the document format generated starting from the models developed during conceptual application design. A context-aware SAF application can be modeled with WebML, using properties[2] that express declaratively the adaptation behavior to be performed by the delivery platform. Declared properties are propagated by the code generation process from the hypertext model to the JSP templates, where they are interpreted by means of proper adaptation rules in the SAF platform during the execution of the application.

Two different approaches can be used to determine the adaptation behavior of the SAF platform:

- *Explicit*: the service designer must explicitly declare the kind of desired adaptation using properties and/or rules.
- *Implicit*: this kind of adaptation is performed automatically by the framework according to a set of predefined rules. For example, if the system knows that the user's current activity is walking/running, the SAF can automatically switch the layout to a one column mode, and, e.g., the font size can be set to a larger size.

Figure 7.2 shows an overview of the architecture of the SAF framework and of its components. The left-hand side of the figure describes

[2] Each WebML construct can be associated with one or more user-defined properties. This feature is also supported by the WebRatio design environment.

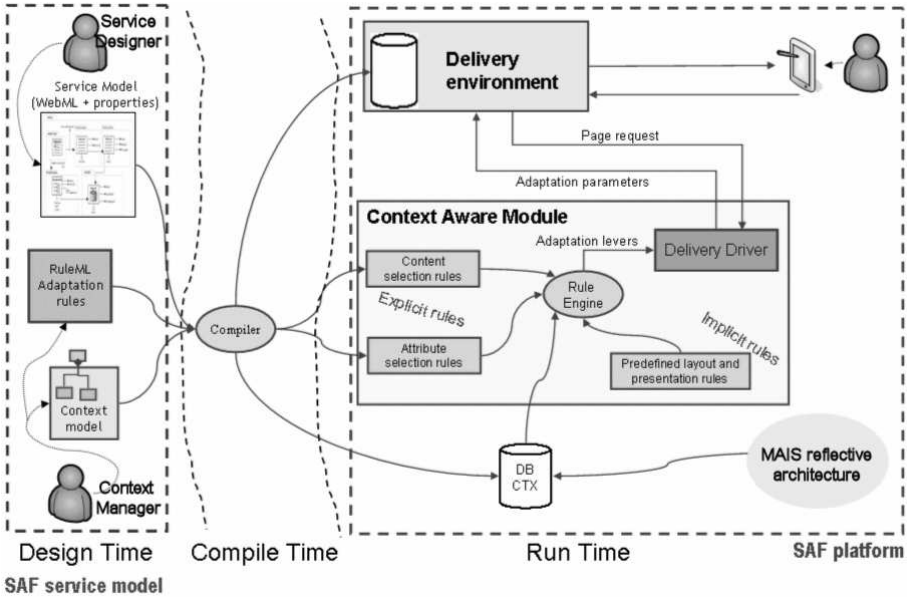


Figure 7.2: The SAF architecture [77].

the *design-time* components. The *context manager* designs the context model and specifies the set of properties and values used to describe the interaction context. The *service designer* models the application using the extended WebML model and explicitly defines the adaptation behavior of the system by tagging adaptive elements (i.e. content units) with proper parameters.

The right-hand side of the figure shows the *run-time* components. The *context-aware module* is responsible for taking decisions about the adaptation actions to be performed. According to the rules that have been defined or that are available as predefined rules and to the state of the context model, the rule engine decides which adaptation actions to perform. Actions are performed by means of a so-called Delivery Driver, used to set the delivery environment configuration, finally performing the actual adaptation of the pages to be delivered to the users.

7.1.2 Multichannel Delivery

Engineering developed the *Dynamic Presentation Manager* (DPM [77, 68]), a software module for adaptive, multichannel delivery of Web applications. While the SAF concentrates on runtime adaptivity of presentation elements, the DPM concentrates on runtime adaptability to different delivery channels (i.e. (X)HTML or WML) and/or device characteristics

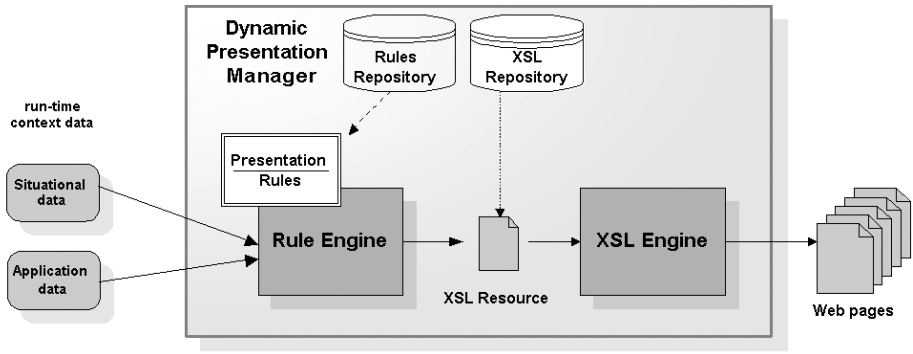


Figure 7.3: The DPM architecture [77].

(i.e. PC, PDA or mobile phone).

In MAIS, the DPM is used to adapt pages designed with WebML and generated by means of the WebML code generator, which produces WebML applications adhering to the MVC (Model View Controller) design pattern. In this context, the DPM module is located in the View layer, leveraging the separation of concerns offered by the MVC pattern.

The actual application data to be published in pages are produced by the application's business logic and depend on the specific application domain. Application data can be adapted by means of the extended WebML environment based on user profile or context data. The DPM does not include the adaptation of contents but concentrates on "look and feel" and layout aspects.

Figure 7.3 shows a functional architecture of the DPM module with its main components. In DPM terminology, context data consists of *situational data* and *application data*, where the three main conceptual entities are: the user profile (the *subject*), hardware and software device features (the *tool*), and application data (the *object*). A typical context state interpreted by the DPM could, for example, be: *screen resolution* $= 1024 \times 768$, *battery level* = high, *memory amount* = 512 MB, and *CPU power* = 1GHz.

The *Rule Engine* component uses proper *presentation rules* to determine the appropriate XSL (eXtensible Stylesheet Language [78]) file to be passed to the *XSL Engine* component. Presentation rules are written using the JESS (Java Expert Shell System [79]) language and are structured as *condition* $\rightarrow$ *action* rules. In DPM, the condition is replaced by a particular context instance and the term action is replaced by the selection of an XSL file. The selection of the appropriate XSL file is based on the context data. The XSL Engine component contains an

XSL transformer, which performs a document transformation using the XSL file selected by the Rule Engine. The starting document contains the application data, serialized in a convenient and transformable form. The result of the transformation is an adapted page to be presented to the final user in an appropriate markup language (i.e. (X)HTML or WML).

The *Rules Repository* contains the presentation rules, and the *XSL Repository* contains the set of available XSL files.

As in the case of SAF, also the DPM uses user-defined properties to be added to WebML hypertext elements, so as to support rule evaluation. After page generation, the properties can then be found in the JSP templates of the View in the MVC architecture. This allows the integration of the DPM into the WebML runtime environment in order to support multichannel content delivery.

An Example Multichannel Delivery

In the context of the MAIS project, the DPM approach yielded a prototype implementation that allowed the verification and test of the described approach by means of a proper example application [70], referring to a tourism scenario. Data, hypertext and presentation model were developed by means of the extended WebML/WebRatio environment, and the application code was generated automatically with the WebML code generator. The system can be accessed from three different kinds of devices: PCs, PDAs, and mobile phones, as showed in Figure 7.4.

The implemented application and the presentation rules for the tourist scenario use an XSL file for each channel/device and support the following adaptation features:

- replacement of widgets (e.g. images, buttons, etc.);
- resizing of page fonts;
- adjustment of the page layout.

Widget replacement is important for improving the usability on different delivery channels and devices with limited hardware resources (i.e. mobile phones). The effects of the replacement are, for example, that images on devices with small screen dimensions are replaced by textual items, and GUI elements such as combo boxes, buttons, and window menus are replaced by lists of items.

Page font resizing is used to present information with varying font sizes. This type of resizing is used to provide easier readable information, for example in the case of disabled users.

Figure 7.4: Multidevice access from various devices: PDA, PC, and mobile phone [77].

Adjustments to the page layout are used to emphasize presentation and customization aspects.

7.1.3 Multimodal Deployment of Adaptive Applications

Cefriel also developed the M^{3}L (MultiModal Markup Language [77, 68]) framework, designed to support the multimodal delivery of contents by synchronizing (in both input and output) a vocal and a visual interaction mode. The two modes were chosen by considering the capabilities and characteristics of the devices available on the market.

The design of the M^{3}L framework is based on the extension of the WebML language. To design multimodal WebML applications, additional information is needed. For every hypertext component of the page it is necessary to specify the interaction modes that can be used, for both input and output. The standard WebML hypertext schemas are thus enriched with properties associated to the single WebML units. Hypertext models are translated into M^{3}L-coded pages, and multimodality properties are translated into specific M^{3}L attributes.

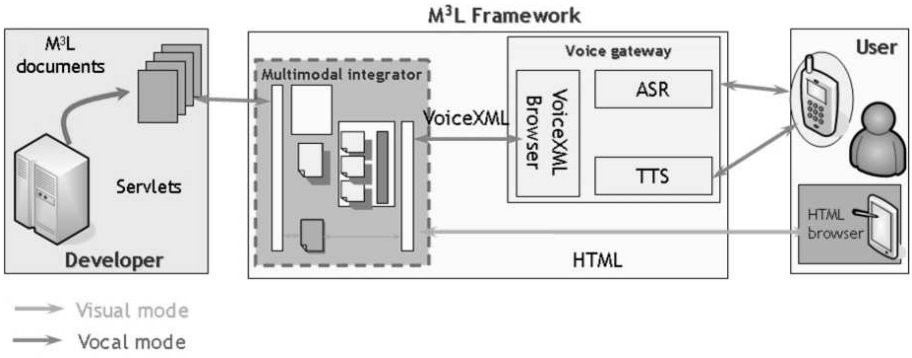


Figure 7.5: M^{3}L framework architecture [77].

The resulting M^{3}L framework is able to manage different devices at the same time, to synchronize them, and to offer a coherent view of the same contents on the two supported channels. By using both channels simultaneously, the user has the impression that he/she is interacting with a single integrated service, even if information is transmitted and delivered through different physical channels. The use of the M^{3}L language requires applications to be coded/generated only once, while at the same time providing the advantages of two communication channels.

The M^{3}L language is defined as a pair of XHTML modules [80]: the *multimodal forms* (used to structure the M^{3}L document) and *M^{3}L forms* (specific to user input) modules. The two modules, together with the XHTML framework, constitute the M^{3}L framework. M^{3}L conforms to the XHTML Host language specification.

The multimodal and M^{3}L form modules define new elements and attributes that enable the management of input (especially for data collected through forms) and output synchronization. *Attributes* allow the developer to choose the best interaction mode for output data, and they allow the developer to select a preferred (or compulsory) input mode. More precisely, the *out* attribute specifies which modes can be used to deliver the content of an element to the user. This attribute is available in any tag that contains content information to be presented to the user. If, for example, the tag `<p>` has its *out* attribute set to `visual`, the text contained is delivered only through the visual mode (i.e. (X)HTML). The *mode* attribute, on the other hand, specifies the modes that a user can use to input data. This attribute is associated with form fields and may have three possible values: `text` to indicate that the user can use a keyboard, `voice` to indicate that the user may use his/her voice, and `all` to say that both input modes may be used.

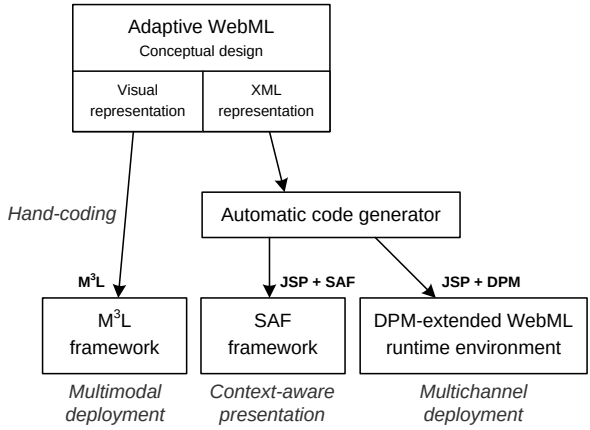


Figure 7.6: Integration of the MAIS development instruments.

Figure 7.5 displays the main components of the architecture of the M^{3}L framework. The *Multimodal integrator* is the core of the multimodal framework. It manages the overall operation logic of the system and integrates the inputs coming from the various connected channels and modes. The integrator determines the outputs to be sent to the user and manages the synchronization between the channels. The *M^{3}L repository* is the container for the multimodal applications and contents delivered through the Multimodal integrator.

The *Voice gateway* is the component that manages the vocal communication between the user and the application. It receives in input VoiceXML documents generated by the Multimodal integrator, interprets them and manages the vocal interaction with the user. A *TTS* (Text-To-Speech) unit is used to generate the voice provided to the user, and an *ASR* (Automatic Speech Recognition) unit is used to manage the user's vocal input. The Voice gateway enables the vocal interaction, allowing the transmission of voice over ordinary PSTN or GSM networks. As an alternative, it is possible to send VoiceXML files directly to the user in those cases where he/she is equipped with a suitable voice browser.

7.1.4 Discussion

Figure 7.6 summarizes the integration of the instruments developed in conjunction with Cefriel and Engineering. Starting point for the resulting design method for multichannel/multimodal adaptive information systems is the context-aware extension of the WebML language, which serves for the conceptual modeling of the application. The SAF and

DPM approaches leverage as well the automatic code generation feature of WebML, while the M^{3}L approach, due to its divergent document format and execution logic, relies on a hand-coded implementation of the hypertext schemas. However, with little more effort, also M^{3}L could be fully integrated into the visual WebRatio environment and the automatic code generator.

The smooth integration of the different research results, even if still at a prototype level, confirmed the viability of the conceived adaptive modeling solutions and of the WebML modeling language.

7.2 Capturing Complex User Behaviors: the Web Behavior Model

Throughout this dissertation, we have seen that there are several techniques that aim at augmenting the efficiency of navigation and content delivery in (Web) applications. Content *personalization*, for example, allows contents and services to be tailored to the users of the application by taking into account predefined roles or proper user profiles. *Adaptive* or *context-aware* Web applications aim at personalizing application contents, layout and/or presentation properties not only with respect to the identity of users, but also by taking into account the context of the interaction involving users and applications. Along a somewhat orthogonal dimension, *workflow-driven* Web applications [81] address the problem of showing the right information at the right time by explicitly modeling the (sometimes hidden) process structure underlying some usage scenarios, especially in business-oriented domains. Eventually, *usability studies* and *Web log analyses* [82] investigate the usability and ergonomics of Web applications by means of an ex-post approach with the purpose of deriving structural weaknesses, checking assumptions made about expected user navigations, and mine unforeseen navigation behaviors for already deployed Web applications.

In this section we describe an extension of the research described in the previous chapters and propose a new approach for triggering application adaptations. More precisely, we combine an adaptive and a process-centric perspective with the aim of supporting the design of *behavior-aware* Web applications , which allow actions to be performed in response to a user's fulfillment of predefined navigation patterns. Such patterns are modeled by means of WBM (*Web Behavior Model*), a simple and intuitive new formalism for specifying navigation goals, and enable the creation of high-level *Event-Condition-Action* rules for expressing novel adaptation requirements. The proposal adopts WebML for hypertext

[1] and adaptation [83] design, but the proposed approach is of general validity and can be applied to arbitrary Web applications.

In this section, we first introduce the Web Behavior Model (Subsection 7.2.1), then we combine it with WebML for defining proper adaptation rules (Subsection 7.2.3). In Subsection 7.2.4 we illustrate a practical example, and in Subsection 7.2.5 we describe a prototype architecture and discuss our experiences gained so far with the prototype implementation. In Subsection 7.2.6, finally, we draw some conclusions for the proposed extension.

7.2.1 The Web Behavior Model

The *Web Behavior Model* (WBM [84, 85]) is a timed state-transition automaton for representing classes of user behaviors on the Web. WBM does not serve for deriving runtime navigation behaviors, but instead allows the description of navigation patterns (at design time) without requiring a profound knowledge of the actual application structure.

Graphically, WBM models are expressed as labeled graphs, providing an easily comprehensible syntax (cf. Figure 7.7). A *state* represents the user's inspection of a specific portion of hypertext (i.e. a page or a collection of pages). State labels are mandatory and correspond to names of pages or page collections. A *transition* represents a navigation from one such portion to another and, thus, the evolution from one state to another. Each WBM specification, called *script*, has at least one initial state, indicated by an incoming unlabeled arc, and at least one accepting state, highlighted by double border lines. Initial states cannot at the same time be accepting states. Each transition from a source to a target state may be labeled with a pair $[t_{min}, t_{max}]$, expressing a time interval within which the transition must occur in order to satisfy the time constraint. Either t_{min} or t_{max} may be missing, indicating open interval boundaries. If a transition does not fire within t_{max} time units, it can no longer occur; on the other hand, navigation actions that occur before t_{min} are lost.

One important aspect of WBM models is that not all navigation alternatives must be covered. As the aim of WBM is to capture a concise set of user interactions, describing particular navigation goals and respective "milestones", even a subset of all possible navigation alternatives may be enough to express the navigation path to be monitored. Typically, Web sites make heavy use of so-called *access pages* that serve the purpose of providing users with browsable categories for retrieving the actual contents offered; also, Web sites usually provide several different access paths toward their core contents. Therefore, by concentrating only on

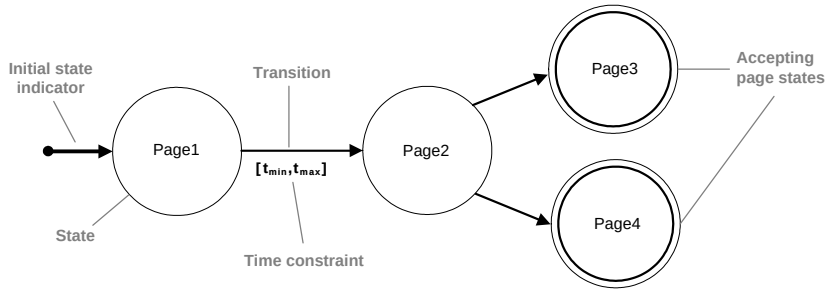


Figure 7.7: Example of WBM script with state, link, time constraints and multiple exiting transitions from one state.

those interactions that really express navigation goals, WBM allows designers both to abstract from unnecessary details and to define small and easily comprehensible specifications. Only performing specified target interactions – in the modeled order – may thus cause transitions in a WBM model.

Figure 7.7 shows an example WBM script. The initial state corresponds to *Page1*. The transition from the first state to the second state occurs only if the user requests *Page2* within t_{min} and t_{max} time units from the moment the script has been initiated, otherwise the script ignores the navigation and remains in its current state. The script in Figure 7.7 also shows two transitions exiting from state *Page2*. The states labeled *Page4* and *Page3* are "competing", as a browsing activity in *Page2* may lead to either *Page4* or *Page3*.

In WBM it is further possible, and sometimes convenient, to use overlapping states, i.e. states corresponding to overlapping portions of hypertext. If two competing states are overlapping, two transitions may trigger simultaneously.

WBM Formal Model

WBM belongs to the class of timed finite state automata. Starting from this class of automata, as discussed in [86, 87], and from the concepts previously introduced, we now give a formal and concise definition of WBM as timed finite state automaton. Step by step we will enrich the formal definition of finite state automata with novel concepts to fully reflect the semantics of WBM.

Definition 14 (Finite State Automaton) *A* finite state automaton *is a tuple* $F = (\Sigma, S, S_0, S_F, E)$*, where:* Σ *is a finite set of input symbols;* S *is a finite, nonempty set of states;* $S_0 \subseteq S$ *is a nonempty set of starting*

states; $S_F \subseteq S$ *is a nonempty set of final states, such that* $S_0 \cap S_F = \emptyset$*;* $E \subseteq S \times S \times \Sigma$ *is a set of* transitions *or* edges.

A timed finite state automaton can be defined as a finite state automaton with transitions constrained in time. Thus, Definition 14 needs to be extended with a clock and the relative definition of time constraints. We here use a discrete-time model; we therefore use the set of nonnegative integer numbers, $\mathbb{N}$, as time domain.

Definition 15 (Time Sequence) *A* time sequence $\tau = \tau_1\tau_2\ldots$ *is an infinite sequence of time values* $\tau_i \in \mathbb{N}$ *with* $i \geq 1$*, satisfying the following constraints:*

1. Monotonicity: $\tau_i < \tau_{i+1}$ *for all* $i \geq 1$.
2. Progress: *For every* $t \in \mathbb{N}$ *there is some* $i \geq 1$ *such that* $\tau_i > t$.

A timed word Σ_t *over an input set of symbols* Σ *is a pair* $\Sigma_t{=}(\sigma, \tau)$ *where* $\sigma = \sigma_1\sigma_2\ldots$ *is an infinite word over* Σ *and* τ *is a time sequence.*

If a timed word $\Sigma_t{=}(\sigma, \tau)$ is used as input to an automaton, then the time τ_i represents the time of the occurrence of the symbol σ_i.

Now, according to [86], we extend the finite state automaton to interpret timed words and associate a clock to the automaton. A *clock* is a variable with values in $\mathbb{N}$. Given that all time values are relative to state transitions, for simplicity in our automaton the clock is reset to zero at each state-transition. The absolute values of time events can be computed by summing the absolute time of the last state transition to the relative time of the given event. Before finally defining the automaton, we also need to formalize time constraints over transitions:

Definition 16 (Time Constraint) *Given a clock* x*, a* time constraint δ *is defined inductively by* $\delta := x \geq c || x \leq c || c_1 \leq x \leq c_2$*, where* $c, c_1, c_2 \in \mathbb{Q}_+$ *and* $c_1 < c_2$.

According to the previous definitions, timed finite state automata can be defined as follows:

Definition 17 (Timed Finite State Automaton) *A* timed finite state automaton *is a tuple* $F = (\Sigma_t, S, S_0, S_F, E, x)$*, where:* Σ_t *is a timed word of input symbols;* S*,*S_0*,*S_F *are defined in Definition 14;* x *is the automaton clock;* $E \subseteq S \times S \times \Sigma_t \times \Phi$ *is the set of transitions, with* Φ *being a set of time constraints* δ *over the clock* x*. An edge* (s, s', σ, δ) *represents a transition from state* s *to state* s' *on input* $\sigma \in \Sigma$ *and subject to the clock constraint* δ *over* x.

Although till now in this dissertation we have been using the term *hypertext* in an intuitive fashion, in order to describe user behaviors by means of a timed finite state automaton, we now introduce a minimal and generic definition of hypertext.

Definition 18 (Hypertext) *A* hypertext *is a couple* $H = (P, L)$ *where* P *is a set of pages and* $L \subseteq P \times P$ *is a set of links, such that every link in* L *connects two and only two pages in* P*.*

Given the definitions above, we can now define the Web Behavior Model (WBM).

Definition 19 (Web Behavior Model) *Given a hypertext* $H = (P, L)$ *and an timed finite state automaton* $F = (\Sigma_t, S, S_0, S_F, E, x)$*, a* Web Behavior Model *is a couple* $WBM(H, F) = (P_l, L_l)$ *where:*

- $P_l : S \rightarrow P \cup P_c \cup \{*\} \cup WBM_i(H, F_i)$ *is a function that maps any state* $s \in S$ *to a page* $p \in P$ *of the hypertext* H*, or to a collection of pages* $P_c \subseteq \mathcal{P}(P)$*, or to the special symbol* $*$ *denoting any page of the hypertext* H*, or to any other Web Behavior Model* $WBM_i(H, F_i)$ *defined over the hypertext* H*.*

- $L_l : T \rightarrow L \cup \{*\}$ *is a function that maps any transition* $e \in E$ *to a link of the hypertext* H*, or to the special symbol* $*$*, denoting an unconstrained navigation between the pages corresponding to the two states connected by the transition* e*.*

This formal definition of WBM is based on the assumption that the automaton does not terminate after receiving unexpected input symbols, and instead keeps its current state, while waiting for a valid input symbol without resetting its clock. Also, as the careful reader may have noticed, WBM models can be described recursively, which allows complex behaviors to be represented by means of sub-models.

7.2.2 WBM and WebML

In this work we use WebML for the modeling of adaptive Web applications. Given the intrinsic semantics of WebML hypertext models, exploiting some structural peculiarities of WebML-based applications allows us to further refine the WBM transition constraints introduced in the previous section. In addition to time constraints, in this section we thus define so-called *state constraints* and *link constraints* to augment the expressive power of WBM models.

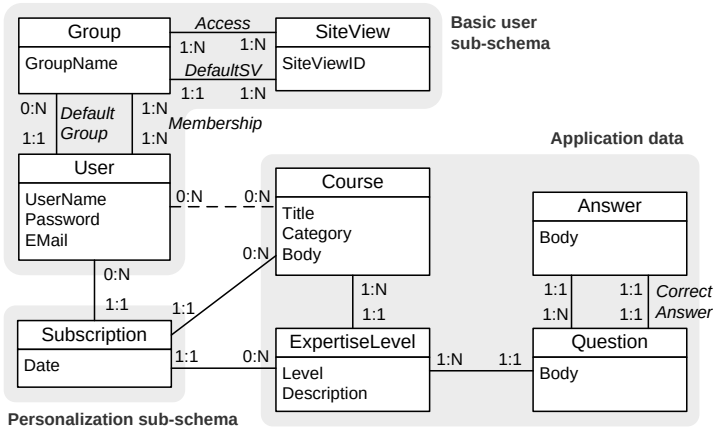


Figure 7.8: An example of Entity-Relationship schema for an e-learning application.

For a better comprehension of the bindings between WBM and WebML, we start with a short description of a reference application scenario, and then we introduce the new concepts.

An E-Learning Application Scenario

Throughout this section we will make use of design examples referring to an e-learning reference scenario introduced in the following. The idea of the application is to provide users with courses (i.e. detailed descriptions of particular subjects) and the possibility to test their knowledge of the respective subjects by means of proper test forms to be filled out. Each course has several knowledge or expertise levels associated, each with a different set of questions, in accordance with the difficulty of the user's ascertained expertise level. Passing a test of a specific course increases the user's respective expertise level. As for now, the reference application does not provide any adaptivity features, which will be introduced in the next subsection.

Figure 7.8 depicts the ER schema underlying the e-learning application: the data schema can be partitioned into the basic *user* sub-schema, a *personalization* sub-schema, and the actual *application data*. Since adaptivity will be triggered through the execution of WBM scripts, the data source does not need any *context* sub-schema.

The personalization sub-schema consists of the entity `Subscription`, which allows the association of users, courses and expertise levels (i.e. the entity represents a ternary relationship). The relationship `User2Course` is derived from the two relationships `User2Subscription` and `Subscrip-`

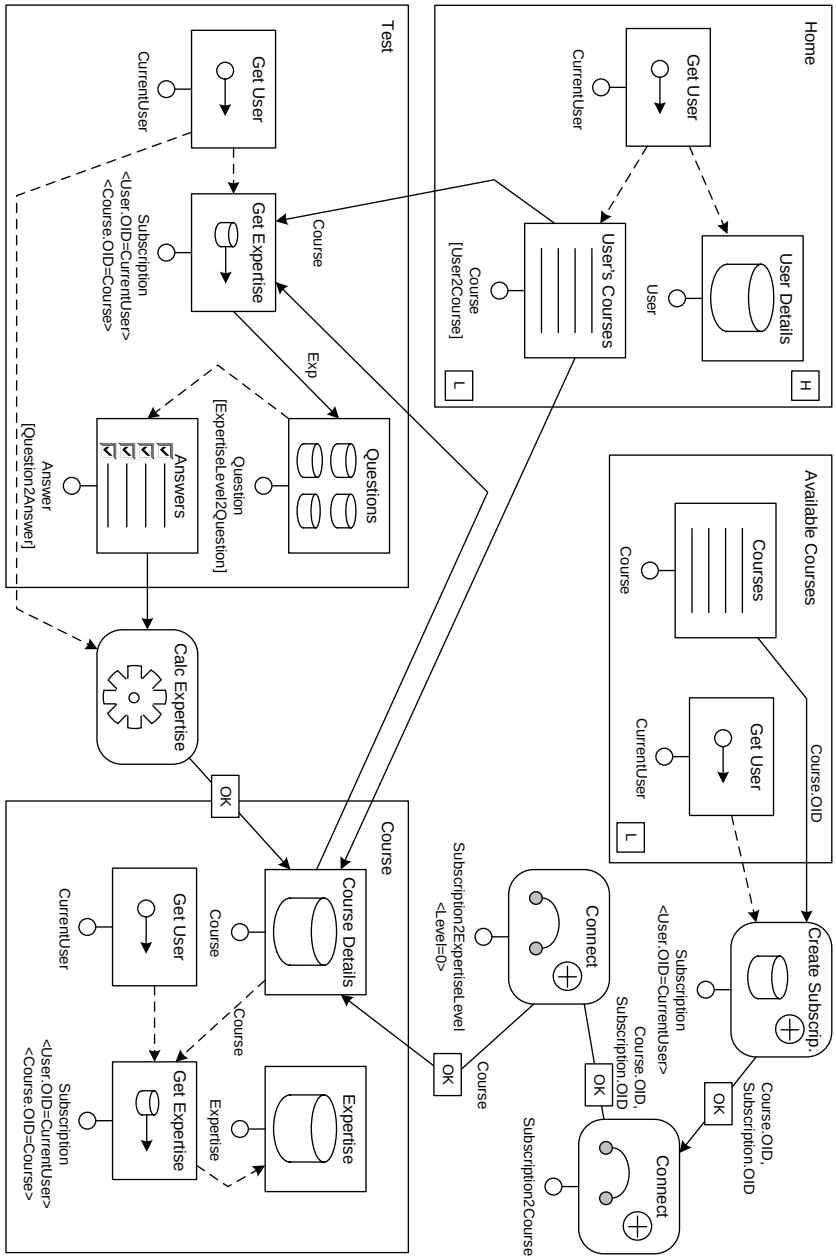


Figure 7.9: The WebML model of the proposed educational Web site.

Figure 7.10: WBM extended set of symbols.

`tion2Course` and provides direct access to the courses a user is subscribed to.

In the core application data we can see that each course may have different expertise levels, which are associated to a set of questions. Questions are associated with a set of answers. The relationship *CorrectAnswer* associates a correct answer to each question.

The WebML hypertext model of the e-learning application is depicted in Figure 7.9. After logging in to the application, the `Home` page publishes personalized data about the user (`User Details` unit) and the list of courses the user is subscribed to (`User's Courses` unit). The `Get User` unit provides the user's identifier for content personalization. By selecting one of the listed courses, the user can either inspect the details of the selected course (i.e. he/she navigates to the `Course` page), or he/she can directly start a test by invoking the `Test` page. But the user can also decide to add a new course to his/her list by accessing the `Available Courses` landmark page, which allows him/her to select a course of interest from the `Courses` index unit. This first creates a new subscription and connects the user with the new course and its lowest level of user expertise, and then forwards the user to the respective course details.

Course details can be browsed in the `Course` page, which also shows the user's current expertise level for the viewed topic (the `Get Expertise` unit gets the respective identifier from the database). From this page, at any time the user can start a test to demonstrate that he/she has increased the knowledge of the topic. Tests are filled out in the `Test` page, which shows a set of questions and possible answers corresponding to the user's expertise level and the selected course. After the submission of a filled answer form, the `Calc Expertise` operation unit computes the (possibly) new level of expertise, updates the underlying database tables and redirects the user back to the `Course` page (through the `OK` link).

Referencing WebML in WBM

WebML hypertext schemas are based on three core elements: areas, pages and units. As shown in Figure 7.10, taking them into account by

means of new WBM state symbols enables easier and more expressive model definitions. Furthermore, contents are published by so-called content units, bound to data entities that can be "queried" to retrieve details about the navigated contents. For this purpose, we also introduce WBM variables, assignments and predicates.

- *Variables* (`<variable name>`) are untyped and named by alphanumerical character sequences, beginning with a character.
- *Assignments* are formulas of the form `<variable name>:=<term>`, where the `<term>` is an arithmetic expression using either constant values, functions, or variables.
- *Predicates* are Boolean expressions of the form `<term1> <comp> <term2>`, where `<term1>` and `<term2>` are arithmetic expressions using either constant values, functions, or variables; and `<comp>` is one of the comparators $=,\leq,\geq,\neq,<$ or $>$. Predicates can be compound to form complex expressions: `<pred> <logicomp> <pred>`, where a `<logicomp>` is one of the two logic operators $\wedge$ (AND), $\vee$ (OR).

To specify *state constraints* over contents and store variable values, four basic *functions* can be used in predicates and assignments:

```
Display(<Data unit name>,<Attribute name>)
Selected(<Data unit name>,<Attribute name>)
Entry(<Entry unit name>,<Form element>)
Parameter(<Operation unit name>,<Attribute name>)
```

The `Display` function applies either to data units, returning the value of an attribute of the displayed entity, or to multi data and index units, returning the set of values of an attribute of the displayed entities. Aggregation functions (such as `SUM, MIN, MAX, AVG, COUNT`) can be applied to sets of values in order to compute scalar values. The `Selected` function applies to index units only, returning the attribute value of the entity that has been selected by the user. The `Entry` function applies to entry units and returns the value of one form field entered by the user. The `Parameter` function applies to operation units, returning the value of one of the parameters assigned in the operation call.

Referring to the `Course` page of our e-learning application, the assignment

```
x := Display(Course, OID)
```

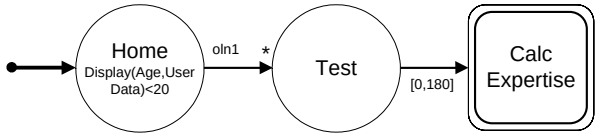


Figure 7.11: A WBM script with an operation, link constraints and predicates.

assigns the `OID` attribute of the item being displayed by the `Course` data unit to the variable `x`. Likewise, the predicate

```
Display(Course, Category) = "Web"
```

verifies whether the `Category` attribute of the item being displayed by the `Course` data unit equals the string "Web" or not. A predicate is satisfied, whenever the expressed condition evaluates to true.

Link constraints are based on WebML link identifiers and allow designers to restrict WBM transitions to explicitly specified links. Link constraints are expressed by labeling transitions with link identifiers. For distinguishing between entering and exiting links, the following syntax can be adopted: a `*` near the begin of a transition arc constrains the link to be an outgoing link; a `*` near the end of a transition arc constrains the link to be an incoming link; if a `*` is specified on both sides of an arc, the relative link must connect directly the two pages or operations; when the `*` is omitted, the transition refers to any path containing the specified link.

The WBM script in Figure 7.11, based on our reference scenario, illustrates the novel concepts. The depicted script aims at identifying newly registered users that are younger than 20 years and take less than 180 seconds to answer the proposed questions. The script starts when entering the `Home` page, displaying a `User Data` unit with `Age` attribute less than 20. The transition to the second state is enabled only if the user reaches the page `Test` through an incoming `OK` link, we suppose the link be denoted by the WebML identifier `oln1`. Finally, the accepting state is reached when the user requests the `Calc Expertise` operation unit within 180 seconds from the activation of the previous state.

WBM has been designed to describe navigation behaviors on top of arbitrary hypertexts. The use of WebML for specifying hypertexts in this section and the introduction of WebML-specific extensions to WBM enable the specification of advanced constraints and to address a finer granularity in the specification of the navigation behavior. However, other conceptual models, providing constructs that express the structure

of the application and the organization of contents, could be adopted as well to specify state and link constraints.

7.2.3 Reacting to User Behaviors

In order to be able to react to observed behaviors and to adapt the running application, we now combine WebML and WBM. For this purpose, we transform the context-triggered adaptivity mechanism introduced Chapter 4 into a WBM-triggered mechanism. Consequently, adaptivity occurs in reaction to the fulfillment of entire WBM scripts, which can be derived from the users' navigation behavior, as outlined in the previous section. According to Subsection 4.3.6, possible reactions in WebML comprise:

- Adaptation of contents published by specific pages.
- Automatic execution of navigation actions toward other pages.
- Adaptation of presentation/style properties.
- Adaptation of the overall hypertext structure.
- Automatic execution of operations or services.

Although independent from one another, expressing adaptation as a combination of WBM scripts and WebML adaptivity primitives intrinsically leads to a high-level ECA paradigm for specifying adaptivity. Commonly, ECA rules respect the general syntax

`on` *event* `if` *condition* `do` *action*

where the event part specifies *when* the rule should be triggered, the condition part assesses whether given *constraints* are satisfied, and the action part states the *actions* to be automatically performed if the condition holds. When specifying behavior-aware Web applications, the event consists of a *page* or *operation request*, the condition requires the fulfillment of a predefined navigation pattern (expressed as *WBM script*), and the action part specifies some adaptivity actions to be forced on the Web application and expressed as a WebML *operation chain*.

While the progression of activated WBM scripts takes into account all navigations performed by a user, the evaluation of entire rules is restricted to a subset of the application's pages. This subset determines the so-called *scope* of the rule and is specified by labeling adaptive pages with an `A`-label in the WebML hypertext model and explicitly associating those pages to the rule's WBM script. Accordingly, events for one

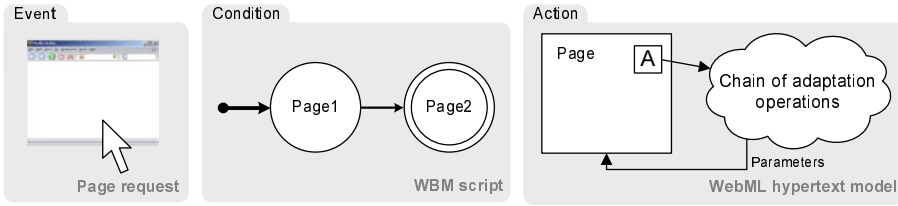


Figure 7.12: High-level ECA rule components.

specific rule are generated only when a user requests a page within the scope of that rule. As a consequence, also the condition of a rule (i.e. the termination of the WBM script) is evaluated only after proper events of the respective rule have been generated. In the case of script termination, the operations indicated by the link exiting the page's `A`-label are executed.

Figure 7.12 graphically summarizes the outlined rule construct: the rule reacts to a user's visit to `Page1` followed by a visit to `Page2` at some stage after his/her visit to `Page1`. The expressed rule condition thus only holds when the script gets to the accepting state `Page2`. Once the accepting state is reached and the user navigates one of the pages within the rule's scope, the operations associated to that page (abstracted as the cloud in Figure 7.12) are executed and possible adaptations may be performed.

Pages may have several competing rules associated and may thus be part of the scopes of different rules. To resolve possible conflicts among concurrently activated rules (each rule may have different associated action parts), proper rule *priorities* enable the selection of the rule with highest priority. Due to the fact that executing the actions of one rule may alter the overall hypertext structure and thus invalidate the semantics of the other simultaneously activated rules, their actions are discarded. Rules are in total priority order, based on explicit numbering or implicit rule creation time.

When considering priorities as properties of rules, rules can be described by the following 4-tuple:

$$\langle \text{Navigation(Scope)}, [\text{WBM Script},] \text{WebML Operations}[, \text{Priority}] \rangle$$

where the *Scope* represents the extent (pages, areas, or site views) within which the rule reacts to navigation events. The *WBM Script* describes the *condition* part of the rule, and the *WebML Operations* specify the *action* part. When the optional *WBM Script* component is omitted, the condition part of the rule always evaluates to true and possible adap-

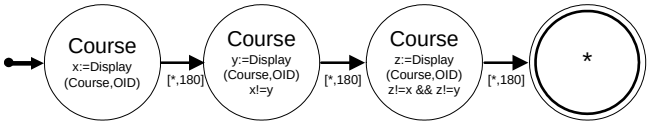


Figure 7.13: A WBM script for triggering the evaluation of a student's knowledge level.

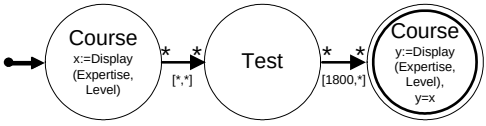


Figure 7.14: A WBM script to penalize unprepared users.

tation operations are executed at page access. Finally, *Priority* is an optional integer expressing the rule's priority with respect to others.

7.2.4 The E-Learning Case Study

By leveraging the described rule model, we now describe two examples that show how proper adaptation operations can be added to the e-learning Web application. Modeled adaptation operations thus extend the already described hypertext model.

Evolving the Level of User Expertise

Figure 7.13 models a WBM script to redirect the user to the `Test` page for the current experience level if he/she has visited 3 different courses (i.e. 3 different instances of `Course` pages), spending less than 3 minutes over each page. Instead of just browsing the topics, the user should be encouraged to read and deepen the presented contents. The `*` in the final state of the WBM script specifies the acceptance of any arbitrary page.

The hypertext model of the e-learning application extended with the necessary adaptivity actions for redirecting the user is depicted in Figure 7.15. The `Get Expertise` unit associated to the `Course` page by means of the link exiting the A-label implements the logic for retrieving the user's current expertise level and for forwarding him/her to the `Test` page. The activating link also provides in input the `Course` parameter, which is an identifier retrieved during page computation. The adaptation may be performed when the user asks for a `Course` page and the script in Figure 7.13 has terminated.

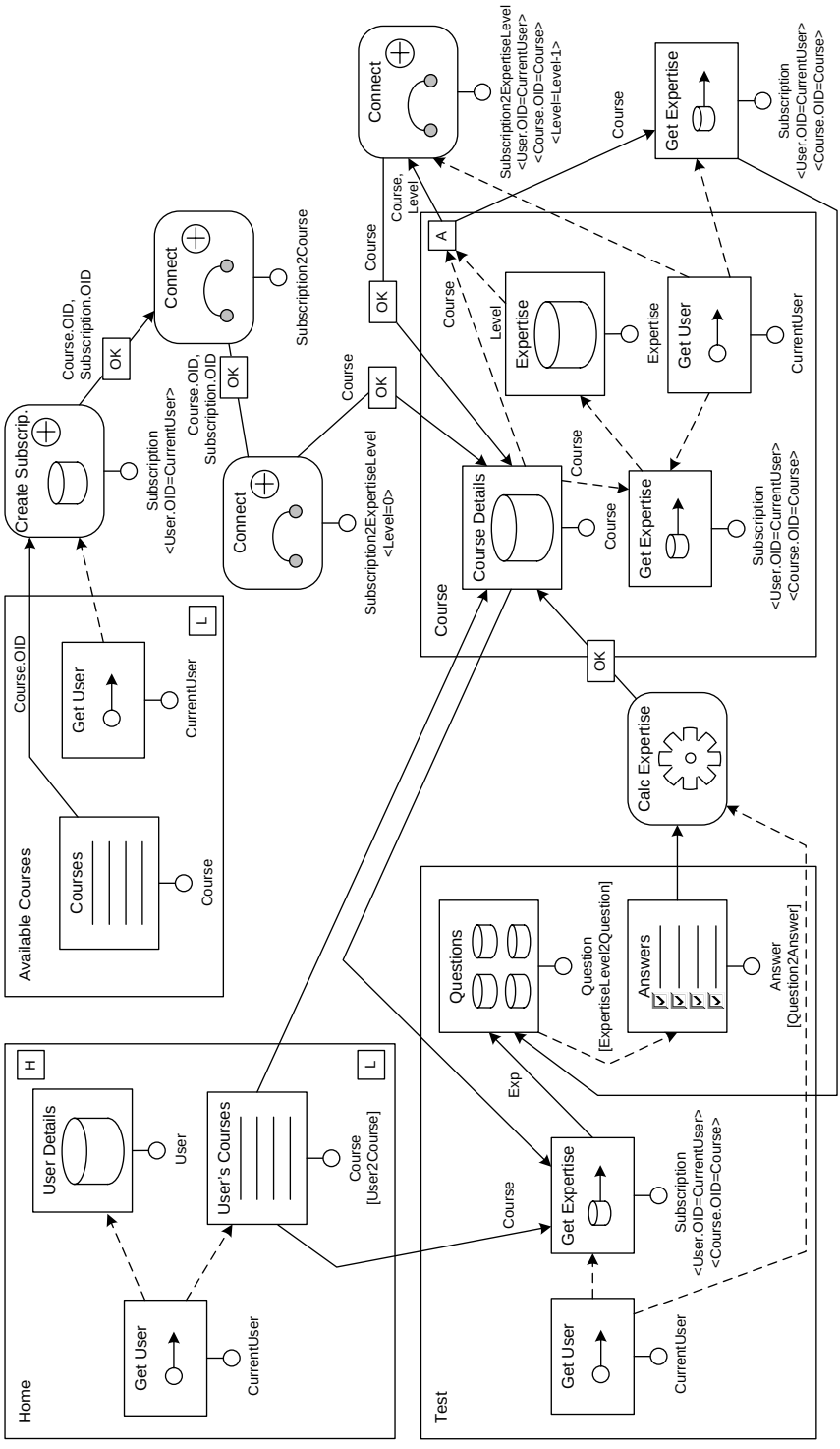


Figure 7.15: The adaptive WebML hypertext model of the e-learning site (cf. Figure 7.9).

Penalizing Slow, Failed Tests

If we suppose that typically test forms do not required more than 15-20 minutes to be filled out by users, we could be interested in penalizing users who take more than 30 minutes (1800 seconds) to fill out a test and, nonetheless, fail. Figure 7.14 shows a possible WBM script for this purpose: starting from the `Course` page, the user needs to go directly to the `Test` page (note the use of the `*` symbols on transitions) and to complete the test in more then 1800 seconds; if the user fails the test, this can be seen in the expertise level of the newly accessed `Course` page, which will not change with respect to its last value. The variables `x` and `y` allow this comparison.

The adaptivity action that penalizes the user is represented by the `Connect` unit in Figure 7.15. The link connecting the A-label of the `Course` page with the `Connect` unit carries two parameters (i.e. `Course` and `Level`) required to perform the association of the user with a lower level of expertise. For the sake of simplicity, we suppose level 0 to be the lowest level of expertise, automatically restrained.

The activating link of this adaptation is performed only if the user accesses the `Course` page and the WBM script in Figure 7.14 has terminated. To prevent possible conflicts with the previously defined adaptivity rule for the `Course` page, we associate a lower priority to the penalizing rule.

7.2.5 System Architecture

In addition to the standard WebML runtime environment, executing behavior-aware Web applications requires proper runtime support for WBM scripts. As illustrated in Figure 7.16, this task is addressed by a suitable *WBM Engine*, which manages WBM scripts based on tracked HTTP requests. More precisely, HTTP requests are automatically forwarded to the WBM engine by the WebML runtime environment, which hosts the actual application. Users interact only with the Web application itself and are not aware of the WBM engine behind it.

The WBM engine collects and evaluates tracked, user-generated HTTP requests for (i) instantiating new scripts at runtime, and (ii) enhancing the states of possible running WBM scripts, as well as (iii) communicating possible script terminations. Script instantiation is managed by a proper *Script loader* module based on tracked page requests and the scripts' starting pages (those indicated by their initial states). The set of scripts that can be instantiated for a particular application is retrieved from a *Script Repository*. A dedicated *WBM Execution Environment*

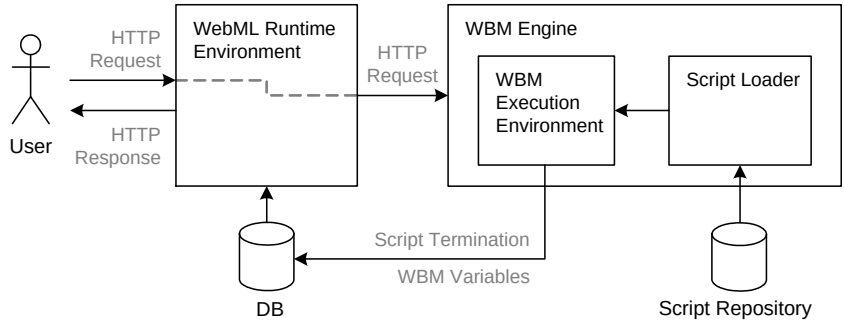


Figure 7.16: Functional architecture of the overall behavior-aware system.

is in charge of progressing instantiated, running scripts. Each user has his/her own set of scripts, which are executed in a completely independent way. Once a script reaches its accepting state, the execution environment communicates the successful termination to the Web application by modifying suitable data structures within the shared database. Those data structures also hold the values of possible WBM variables used during script execution. After successful termination of a WBM script, the Web application possesses all the necessary data for executing the possibly associated adaptation actions.

As soon as a user requests one of the pages within the scope of the high-level rule whose condition is satisfied by the terminated WBM script, the Web application executes the adaptation operations associated to the requested page (cf. the extended page computation logic for context-aware pages summarized in Figure 4.10). For this purpose, page computation starts by checking whether scripts connected to the page have terminated or not, before proceeding with the actual rendering of the page. If there are terminated scripts for that page, one or more rules could be executed. In case of multiple candidate rules, rule priorities help determining the rule with highest priority. Thus, computation proceeds with the determined adaptation operations, producing effects as described in Subsection 7.2.3. Only afterward and in the case of no automatic navigation actions, computation continues with the actual page and a suitable HTML response is produced.

Adaptivity Policies

According to the adaptivity policies introduced in Subsection 4.3.5, the WBM engine supports a *synchronous* and an *asynchronous* rule execution model.

In the *synchronous* case, the page request is immediately forwarded to the WBM engine when a user requests a page inside the scope of a rule, and page computation proceeds only after receiving a respective status notification from the WBM engine. In this way, possible WBM scripts are updated before page computation, and possible script terminations are enabled to immediately cause an execution of adaptivity actions. This, however, implies that the speed of page computations heavily depends on the performance of the WBM engine.

When adopting an *asynchronous* configuration, page requests are satisfied immediately and forwarded only after page computation. Possible adaptations are performed after the predefined refresh/polling interval by automatically refreshing the page or by means of the background monitoring mechanism described in Section 5.3. The asynchronous configuration allows a better parallelization of the Web application and the WBM engine, as well as short response times.

WBM Engine Implementation

The implementation of rule engines for active databases is a well known and studied topic in the literature on database systems. Our problem of handling user sessions and WBM scripts resembles to the problem of handling transactions and rules in active databases, wherefore the implementation of the WBM Engine [84, 85] has been inspired by the Starbust Active Database [65].

More precisely, the developed rule engine maintains a catalog of WBM scripts, where each script has associated the collection of user sessions that activated the script. As the number of instances of user sessions in a heavily used Web application is much bigger than the number of defined WBM scripts, it is more efficient to associate user sessions to scripts than running scripts to each single user session. This reduces the effort needed to maintain the data structures in memory and improves the overall system performance. The WBM engine thus inserts tokens representing single user sessions into the scripts' runtime data structures. Tokens advance according to script state changes and are removed when entering an accepting state. Removing a token implies communicating the successful termination of the respective script to the database, thus the Web application.

However, user sessions are not permanent objects. Due to the HTTP protocol, it is not possible to establish when exactly a user session terminates. This may cause a high number of running WBM scripts potentially never reaching an accepting state. To solve this problem and to handle WBM time constraints, the WBM Engine makes use of a garbage

collector, which performs cyclic checks and clean-ups of activated scripts with inactive sessions, where inactivity implies no user navigations for a predefined interval of time.

In order to guarantee the consistency of user click streams, tracked page requests are stored in a chronologically ordered queue. The described tracking process thus requires in input the complete URL navigated by the user, the timestamp of the request, an identifier of the user (e.g. the session identifier), as well as the identifier of the Web application itself[3].

Prototype Experiments

A first prototype of the presented WBM engine has been developed and tested by implementing the behavior-aware e-learning Web application used throughout this section. We have fully implemented link and time constraints as well as most of the mechanisms required by WBM state constraints. Results from experiments are quite positive: experiments proved the viability of the approach, and the use of the asynchronous execution model effectively avoids WBM engine response times to impact too much on the user experience.

Experiments, however, revealed a performance problem to be solved in the next version of the prototype: we observed an excessive lag between the start of a notification of a page request and the final computation of the new state by the WBM engine (around 2.5 seconds to manage 100 user requests[4]). Further studies proved that the bottleneck of the system was not the WBM engine (a stand-alone version of the engine processes the same 100 requests in less than 60 milliseconds, and can efficiently support WBM scripts of greater complexity than the ones described in this section). Rather, the bottleneck could be detected in the generation of SOAP messages by the Web application, as the communication between the Web application and the WBM engine is currently implemented by means of Web services.

7.2.6 Discussion

In this section we have proposed a further extension to the modeling of adaptive or context-aware Web applications, i.e. a general purpose approach for building behavior-aware Web applications. Our proposal

[3] This information could be reconstructed as well from the requested URL.

[4] Experiments were realized using an AMD AthlonXP 1800+, 512MB of RAM and with Tomcat as Web server.

combines the context-aware extension of WebML and WBM into a visual ECA paradigm that allows one to associate the adaptive features described in the previous chapters to structured navigation patterns.

When combined, WebML and WBM yield a very powerful model, with adequate expressive power to capture highly sophisticated Web dynamics. WBM enables the monitoring of behaviors ranging from rather coarse to very detailed event sequences; the binding of WBM predicates to WebML enables the specification of events in terms of hypertext elements (pages and links) and of the application content. While the combined expressive power is quite strong, we also believe that the two models should be kept distinct, so as to enable the specification of WebML applications that are totally independent from WBM and of WBM scripts that can be progressively refined and finally bound to WebML concepts.

7.3 Enabling Runtime Adaptivity Management

As is the nature of the Web engineering discipline, the conceptual modeling approaches to adaptability, context-awareness, and adaptivity primarily focus on the definition of *design processes* to achieve adaptation, thereby providing efficient methods and tools for the design of such a class of applications. For instance, model-driven methods [8, 88], object-oriented approaches [43], aspect-oriented approaches [46], and rule-based paradigms [44, 89] have been proposed for the specification of adaptation features in the development of adaptive Web applications (see also Section 2.4 for a detailed discussion of the approaches). The resulting specifications facilitate the implementation of the adaptation requirements and may also enhance code coherence and readability. Unfortunately, in most cases, during the implementation phase all formalizations of adaptivity requirements are *lost*, and the adaptivity features become buried in the application code. This aspect implies, that changes and evolutions of adaptive behaviors after the deployment of the application are difficult, unless a new version of the application is implemented and released.

Based on the experience gained with the model-driven design of adaptive and/or context-aware Web applications as described in this dissertation, we believe that another interesting area of investigation in this research field is the support of the *dynamic* management of adaptivity features: on one hand, this will require proper design time support (e.g. languages or models), on the other hand, this will require suitable runtime environments, where adaptivity specifications can be easily administered.

In [90] we outlined our first ideas on this topic and described a possible conceptual framework for the approach. In this section we focus on the evolution of that work, describing in more detail a rule-based language (ECA-Web) for the specification of adaptive behaviors, orthogonally to the application design, and its concrete implementation. The resulting framework provides application designers with the ECA-Web language and application administrators with the possibility to easily manage ECA-Web rules (inserting, dropping, and modifying rules), even after the implementation and the deployment of the application, i.e. at runtime. As envisioned above, with the described approach we enable the decoupled management of adaptivity features at both design time and runtime.

This section is organized as follows. Section 7.3.1 introduces the ECA-Web rule language for the specification of adaptive behaviors for Web applications and, then, shows how ECA-Web rules can be executed by a proper rule engine and integrated with the execution environment of the adaptive Web application. Section 7.3.2 discusses a prototype of adaptive Web application supported by ECA-Web rules and shows the usage of the active rule language. Section 7.3.3 describes the implementation of the overall system and reports on first experiences with the rule-based adaptivity specification and the runtime management of adaptivity rules. Finally, Section 7.3.4 concludes the section.

7.3.1 Enabling Dynamic Adaptivity Management

In following we introduce the *design* component (the ECA-Web language) and the *runtime* component (the rule execution environment) that enable the dynamic administration of adaptivity features.

ECA-Web

ECA-Web is an XML-based language for the specification of active rules, conceived to manage adaptivity in Web applications. The syntax of the language is inspired by Chimera-Exception, an active rule language for the specification of expected exceptions in workflow management systems [91]. ECA-Web is an evolution of the Chimera-Web language we already proposed in [90], and it is equipped with a proper rule engine for rule evaluation and execution.

The general structure of an ECA-Web rule is summarized in Figure 7.17. A typical ECA-Web rule is composed of five parts: scope, events, conditions, action, and priority. The *scope* defines the binding of the rule with individual hypertext elements (e.g. pages, links, contents

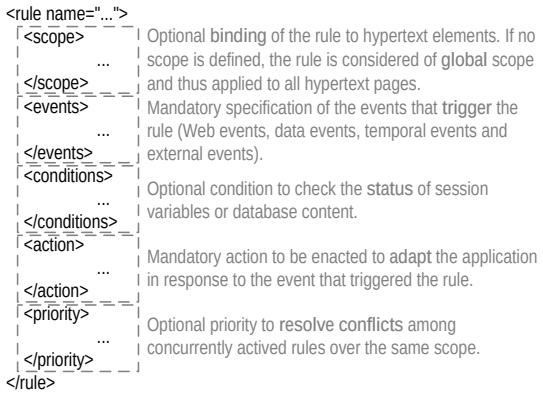


Figure 7.17: Structure of ECA-Web rules.

inside pages). By means of *events* we specify how the rule is triggered in response to user navigations or changes in the underlying context model. In the *condition* part it is possible to evaluate the state of application data (e.g. database contents or session variables) to decide whether the action is to be executed or not. The *action* specifies the adaptation of the application in response to a triggered event and a true condition. The *priority* defines an execution order for rules concurrently activated over the same scope; if not specified, a default priority value is assigned. More details on the rule specification by means of ECA-Web are given in the next section, where we discuss the architecture of the runtime environment for rule execution. An example of ECA-Web rule will then be shown in Subsection 7.3.2.

The Integrated Runtime Architecture

The execution of ECA-Web rules demands for a proper runtime support. Figure 7.18 summarizes the functional architecture of the system, highlighting the two main actors: the *Rule Engine* and the *Web Server* hosting the Web application. The *Rule Engine* is equipped with a set of *Event Managers* to capture events, and a set of *Action Enactors* to enable the execution of actions. The communications among the single modules are achieved through asynchronous message exchanges (*Message-Oriented Middleware*).

Event Managers. Each type of ECA-Web event is supported by a suitable event manager (i.e., *Web Event Manager*, *Data Event Manager*, *Temporal Event Manager*, and *External Event Manager*). Event managers and ECA-Web provide support for the following event types:

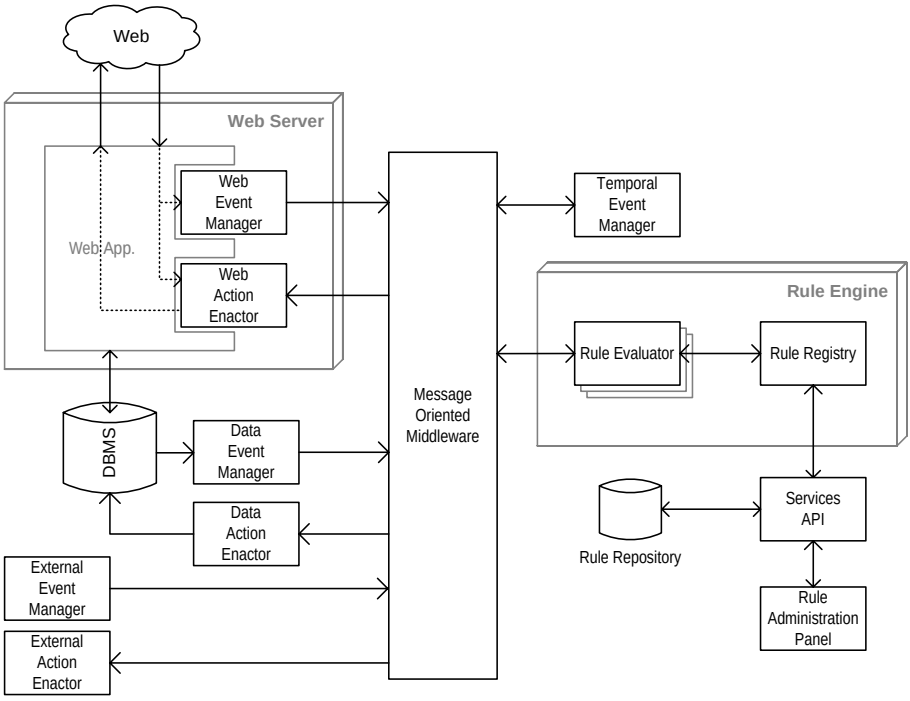


Figure 7.18: Functional architecture of the integrated execution environment for adaptive Web applications.

- *Data events* refer to operations on the application's data source, such as `create`, `modify`, and `delete`. In adaptive Web applications, such events can be monitored on user, customization, and context data to trigger adaptivity actions with respect to users and their context of use. Data events are managed by the *Data Event Manager*, which runs on top of the application's data source.

- *Web events* refer to general browsing activities (e.g. the access to a page, the submission of a form, the refresh of a page, the download of a resource), or to events generated by the Web application itself (e.g. the start or end of an operation, a login or logout of the user). Web events are risen in collaboration with the Web application and captured by the *Web Event Manager*. Since adaptivity actions are typically performed for each user individually, Web events are also provided with a suitable user identifier.

- *External events* can be configured by a dedicated plug-in mechanism in form of a Web service that can be called by whatever application or resource from the Web. When an external event

occurs, the name of the triggering event, and suitable parameters are forwarded to the rule engine. External events are captured by means of the *External Event Manager*.

- *Temporal events* are subdivided into *instant*, *periodic*, and *interval events*. Interval events are particularly powerful, since they allow the binding of a time interval to another event (*anchor event*). For example, the expression "five minutes after the access to page X" represents a temporal event that is raised after the expiration of 5 minutes from the *anchor event* "access to page X". Temporal events are managed by the *Temporal Event Manager*, based on interrupts and the system clock.

The managers for external and temporal events are general in nature and easily reusable. The *Data Event Manager* is database-dependent[5]. The *Web Event Manager* requires a tight integration with the Web application.

Action Enactors. *Actions* correspond to modifications to the Web application or to executions of back-end operations. Typical adaptation actions are: adaptation of page contents, automatic navigation actions, adaptation/restructuring of the hypertext structure, adaptation of presentation properties, automatic invocation of operations or services. Adaptations are performed according to the user's profile or his/her context data.

While some actions can easily be implemented without any explicit support from the Web application (e.g. the adaptation of page contents may just require the setting of suitable page parameters when accessing the page), others may require a tighter integration into the application's runtime environment (e.g. the restructuring of the hypertext organization). The level of application support required for the implementation of the adaptivity actions thus heavily depends on the actual adaptivity requirements. However, application-specific actions can easily be integrated into the ECA-Web rule logic and do not require the extension of the syntax of the rule language (an example of the use of actions is shown in Figure 7.23).

As depicted in Figure 7.18, the execution of adaptivity actions is performed by means of three action enactors: *Web Action Enactor*, *External Action Enactor*, and *Data Action Enactor*. Web actions need to be provided by the application developer in terms of Java classes; they

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

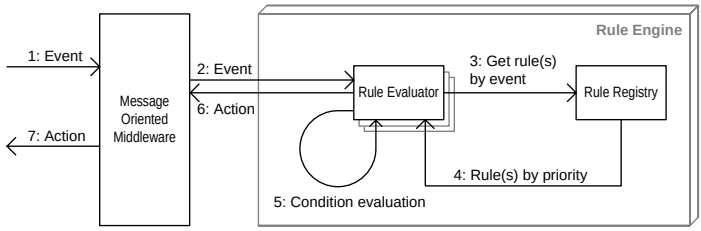


Figure 7.19: The rule engine: internal rule execution logic.

are performed by the *Web Action Enactor*, which is integrating into the application's runtime environment, in order to guarantee access to the application logic. External actions are enacted through a dedicated Web service interface. Data actions are performed on the database that hosts the application's data source.

The enactor for external actions is general in nature and easily reusable, the *Data Action Enactor* is database-dependent, the *Web Action Enactor* is integrated with the Web application.

Rule Engine. In the architecture depicted in Figure 7.18, the *Rule Engine* is in charge of identifying the ECA-Web rules that correspond to captured events, of evaluating conditions, and of invoking action enactors – in case of true conditions.

In the rule engine, a scalable, multithreaded *Rule Evaluator* evaluates conditions to determine whether the rule's action is to be performed or not, depending on the current state of the application. In ECA-Web, *conditions* consist of predicates over context data, application data, global session variables, and/or page parameters. For example, in the condition part of an ECA-Web rule it is possible to specify parametric queries over the application's data source, where parameters can be filled with values coming from session variables or page parameters.

The rule engine also includes a *Rule Registry* for the management of running, deployed ECA-Web rules. Deployed rules are loaded into the *Rule Registry*, a look-up table for the efficient retrieval of running rules, starting from captured events. The internal execution logic of a triggered rule is graphically summarized in Figure 7.19.

ECA-Web Rule Management

While the *Rule Registry* contains only deployed rules for execution, the *Rule Repository* offers support for the persistent storage of rules. For the management of both *Rule Registry* and *Rule Repository*, we provide

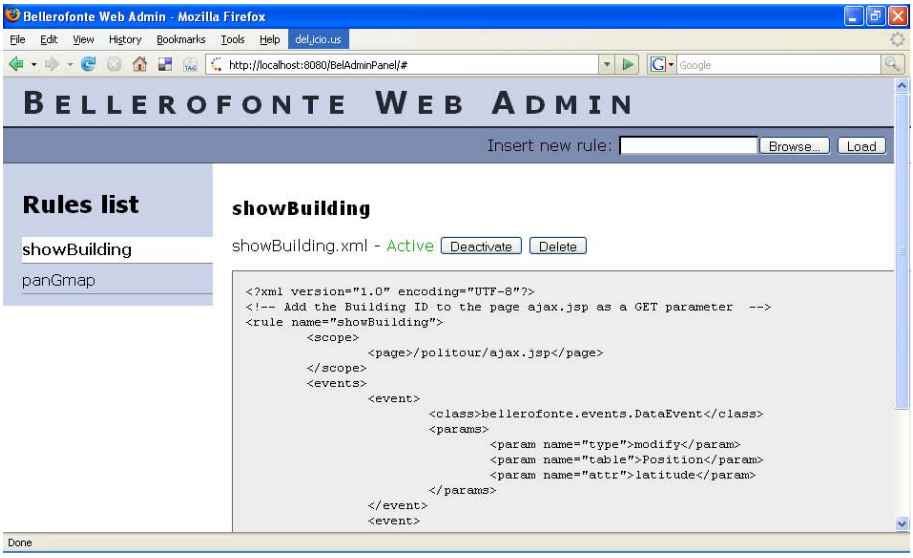


Figure 7.20: The current Web interface for the runtime rule management.

a *Rule Administration Panel* that allows application designers to easily view, add, remove, activate, and deactivate rules. Figure 7.20 shows a screenshot of the *Rule Administration Panel*.

Deploying ECA-Web Rules

Activating or deploying an ECA-Web rule is not a trivial task and, depending on the rule specification, may require to set up a different number of modules. During the deployment of an ECA-Web rule, the XML representation of the rule is decomposed into its constituent parts, i.e. scope, events, conditions, action, and priority, which are then individually analyzed to configure the system. The scope is used to configure the *Web Event Manager* and the *Web Action Enactor*. The events are interpreted to configure the respective event managers and to set suitable triggers in the application's data source. The conditions are transformed into executable, parametric queries in the *Rule Registry*. The action specification and the rule's priority are as well fed into the *Rule Registry*. Each active rule in the system is thus represented by an instance in the *Rule Registry*, (possibly) by a set of data triggers in the database, and by a set of configurations of the event managers and the action enactors.

The registry allows the concurrent access by multiple *Rule Evaluators*. Priorities are taken into account in the action enactor modules, which

select the action to be performed for the page under computation (the scope) from the queue of possible actions, based on rule priorities.

During the deployment of an ECA-Web rule, conflict resolution and termination analyses will be performed in line with the methods conceived and implemented for the Chimera-Exception language [91].

Enacting Adaptivity

External and *data* actions can be executed immediately upon reception of the respective instruction from the rule engine. The enaction of *Web* actions, which are characterized by adaptations visible on the user's browser, is possible only when a "request for adaptation" (a page request) comes from the browser. In fact, only in presence of an explicit page request, the Web application is actually in execution and, thus, capable to apply adaptations. This is due to the lack of suitable push mechanisms in the standard HTTP protocol.

In order to provide the application with active/reactive behaviors, in our previous works we therefore studied two possible solutions: (i) periodically *refreshing* the adaptive page currently viewed by the user [88], and (ii) periodically monitoring the execution context in the background (e.g. by means of suitable Rich Internet Application – RIA – technologies) and refreshing the adaptive page only in the case adaptivity actions are to be performed [92, 90]. Both mechanisms are compatible with the new rule-based architecture and enable the application to apply possible adaptivity actions that have been forwarded to the *Web Action Enactor* by the *Rule Engine*.

7.3.2 Case Study

As a case study, in the following we re-consider the PoliTour application already discussed in Chapter 6. More precisely, in the following we use the WebML notation for two distinct purposes: (i) to easily and intuitively describe the reference application, and (ii) to highlight how the active rules to be introduced may take advantage of a formally defined, conceptual application model for the definition of expressive adaptivity rules. The approach we propose in the following, however, is not tightly coupled to WebML and can be used in the context of any modeling methodology upon suitable adaptation.

It is worth noting that the approach based on ECA-Web described in this section is not to be considered an *alternative* solution to the conceptual design approaches so far proposed in the literature for Web application modeling. Rather, we believe that the best expressiveness

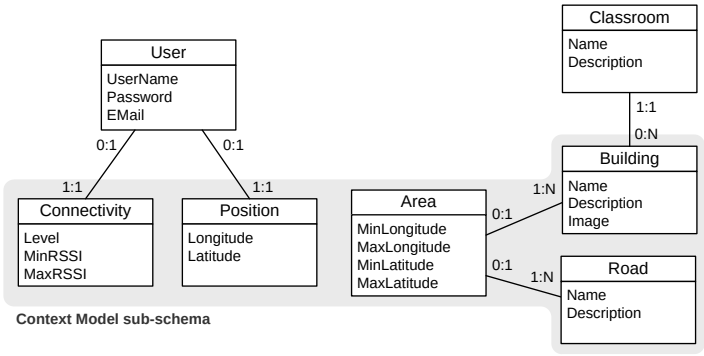


Figure 7.21: Relational data model of the PoliTour application.

and a good level of abstraction for the illustrated adaptivity specification language will be achieved by *complementing* the current modeling and design methods (such as WebML, Hera, OO-H, or OOHDM). In fact, in this section we hint at the specification of ECA-Web rules on top of WebML (both data and hypertext models), just like SQL triggers are defined on top of relational data models. This consideration is in line with the proposal by Garrigós et. al [89], who show how to apply their PRML rule language to several different conceptual Web application models.

The conceptual model of the application serves as terminological and structural reference model for the specification of adaptivity rules and, thus, allows application developers to keep the same level of abstraction and concepts already used for the design of the main application. In terms of WebML, for example, this could mean to restrict the scope of individual rules to specific hypertext elements like *content units*, *pages*, or *areas*, or to relate events to specific *links* or *units*. The same holds for actions, which could for example be applied to single *units* or even *attributes*.

Application Design with WebML

Figure 7.21 depicts a simplified version of the data schema underlying the PoliTour application. Five entities compose the context model, upon which the context-aware features of the application are specified. The entities `Connectivity` and `Position` are directly connected to the entity `User`, as they represent context data which are individual for each user of the system. `Position` contains the latest GPS coordinates for each user, `Connectivity` contains a set of discrete connectivity levels that can be associated to users, based on their current RSSI. GPS coordinates and RSSI are sensed at the client side and periodically communicated to the

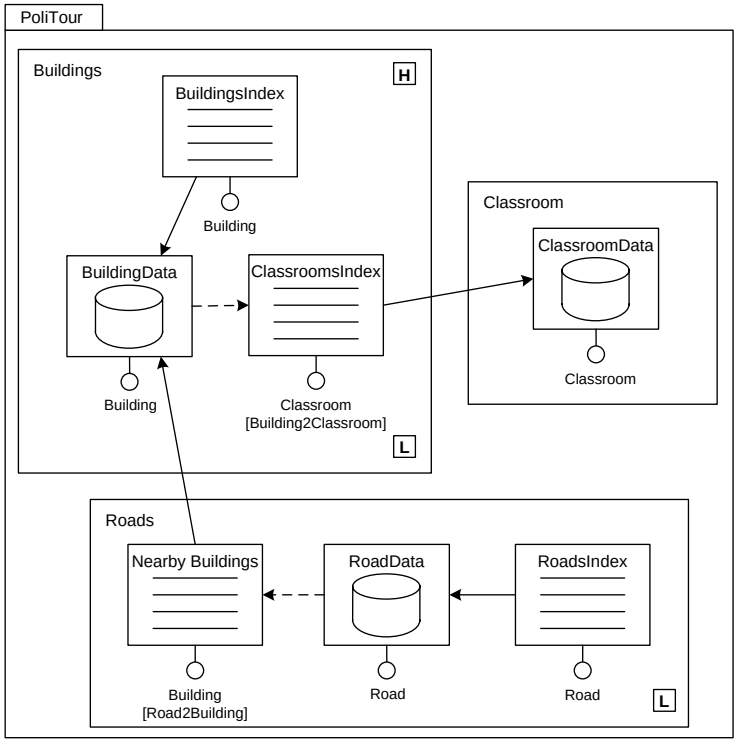


Figure 7.22: Simplified hypertext model of the PoliTour application. `H` stands for `Home` page; `L` stands for `Landmark` page.

application server in the background [92]. The entities `Area`, `Building`, and `Road` provide a logical abstraction of raw position data: buildings and roads are mapped onto a set of geographical areas inside the university campus, which enables the association of a user with the building or road he/she is located in based on the GPS position. The entity `Classroom` is located outside the context model, as the application is not able to react to that kind of granularity, and the respective data is considered additional application content.

For a better comprehension of the following discussion, Figure 7.22 re-proposes the non-adaptive hypertext schema of Figure 6.3, already described in Subsection 6.1.2. At this point, we just recall the two adaptivity requirements posed to the application: (i) provide users with location-aware information on buildings and roads inside the university campus, and (ii) alert users about low connectivity conditions.

Defining an ECA-Web Rule

The full specification of the application's adaptivity requires several different ECA-Web rules to manage the adaptation of the contents in the pages `Buildings` and `Roads`, and to alert the user of low connectivity conditions. Figure 7.23 shows the ECA-Web rule that adapts the content of the page `Buildings` to the position of the user inside the university campus.

The scope of the rule binds the rule to the `Buildings` page. The triggering part of the rule consists of two data events, one monitoring modifications to the user's `longitude` parameter, one monitoring the user's `latitude` parameter. In the condition part of the rule, we check whether there is a suitable building associated to the user's current position (`<notnull>` condition), in which case we enact the `Showpage` adaptivity action with new page parameters, suitably computed at runtime; otherwise, no action is performed. In the action part of the rule we link the `bellerofonte.actions.Showpage`[6] Java class, which contains the necessary logic for the content adaptation action. The variable `building_oid` has been computed in the condition part of the rule and is here used to construct the URL query to be attached to the automatic page request that will cause the re-computation of the page and, thus, the adaptation of the shown content.

It is worth noting that the scope of the previous rule is limited to one specific hypertext page (the `Buildings` page). There might be situations requiring a larger scope. For example, the rule for alerting users about low connectivity is characterized by a scope that spans all the application's pages; in terms of WebML, binding an ECA-Web rule to all pages means to set the scope of the rule to the site view, i.e. a model element (see site view `PoliTour` in Figure 7.22). The scope of the rule is specified as follows:

```
<scope>
  <siteview>PoliTour</siteview>
</scope>
```

As for the dynamic management of adaptivity rules, we could for example be interested in testing the two adaptivity features (location-aware contents and the low connectivity alert) independently. We would thus first only deploy the rule(s) necessary to update the contents of the `Buildings` and `Roads` pages and test their functionality without also enabling the alert. Then we could disable this set of rules and enable

[6] *Bellerofonte* is the current code name of the rule engine project.

```
<rule name="showBuilding">
  <scope>
    <page>/politour/building.jsp</page>
  </scope>
  <events>
    <event>
      <class>bellerofonte.events.DataEvent</class>
      <params>
        <param name="type">modify</param>
        <param name="table">Position</param>
        <param name="attr">latitude</param>
      </params>
    </event>
    ...
  </events>
  <conditions>
    <object>
      <name>P</name>
      <type>Position</type>
      <requirements>
        <eq><value>user_id</value><value>Rule.currentUser</value></eq>
      </requirements>
    </object>
    <object>
      <name>A</name>
      <type>Area</type>
      <requirements>
        <lt><value>MinLatitude</value><value>P.Latitude</value></lt>
        <gt><value>MaxLatitude</value><value>P.Latitude</value></gt>
        <lt><value>MinLongitude</value><value>P.Longitude</value></lt>
        <gt><value>MaxLongitude</value><value>P.Longitude</value></gt>
      </requirements>
    </object>
    <notnull>
      <value>A.building_id</value>
    </notnull>
  </conditions>
  <action>
    <class>bellerofonte.actions.Showpage</class>
    <params>
      <param name="redirectURI">building.jsp?id=<value>building_id</value></param>
    </params>
  </action>
</rule>
```

Binding of the rule to the *Building* page

The rule may be triggered by two data events, i.e. the modification of the current user's *latitude* or *longitude*. For presentation purposes, we only show the event related to the *latitude* parameter.

The specification of the rule's condition requires the definition of two data objects for the construction of the database query: the first one (*P*) extracts the current user's position by means of the *Rule.currentUser* environment variable; the second one (*A*) extracts the area associated to the user's current position. Finally, the *<notnull>* condition allows us to check the presence of a building in the identified area.

The adaptation of the page contents requires the invocation of the *Showpage* action with suitable parameters computed at runtime.

Figure 7.23: The ECA-Web rule for checking the user's current position and updating page contents.

the rule for the alert and test it. If both tests are successful, we finally could enable both adaptivity features in parallel and test their concurrent execution.

7.3.3 Implementation

The proposed solution has been developed with scalability and efficiency in mind. The Web application and the rule engine are completely decoupled, and all communications are based on asynchronous message exchanges based on JMS (Java Message Service). The different modules of the proposed system can easily be distributed over several server machines. The overhead introduced into the Web application is reduced to a minimum and only consists of (i) forwarding Web events and (ii) executing adaptivity actions. These two activities in fact require access to the application logic. In fact, depending on the required adaptivity support, event mangers and action enactors may require different levels of customization by the Web application developer. The customization consists in the implementation of the application-specific events and of the actions that are to be supported by the adaptive application.

In order to perform our first experiments with ECA-Web and the rule engine, we have adapted the PoliTour application, which we already extensively tested when developing our model-driven approach to the design of context-aware Web applications [92]. As for now, our experiments with a limited number of rules have yielded promising results. Experimentations with larger numbers of active rules, different adaptive Web applications, and several users in parallel are planned.

Also, to really be able to take full advantage of the flexibility provided by the decoupled adaptivity rule management, a set of suitable adaptivity actions needs to be implemented. Our current implementation provides support for data actions and a limited set of Web actions (namely, `ShowPage` for adapting page contents, and `ChangeStyle` for adapting presentation style properties). Data actions are currently applied only to entities and attributes that are directly related to the user for which the action is being executed. Also, condition evaluation is automatically confined to those context entities and attributes that are related to the user for which the rule is being evaluated. We are already working on extending condition evaluation to any application data, coming from the data source as well as from page and session parameters.

In the context of WebML, the provision of a set of predefined adaptivity actions will lead to a library of adaptivity actions, possibly integrated into the WebML runtime environment. In the case of general Web applications, the rule engine can be used in the same fashion and with the

same flexibility, provided that implementations of the required adaptivity actions are supplied.

7.3.4 Discussion

We believe that the decoupled runtime management of adaptivity features represents an important area of investigation in adaptive Web applications. In this section we have therefore shown how to empower design methods for adaptivity with the flexibility provided by a decoupled environment for the execution and the administration of adaptivity rules.

The development of Web applications in general is more and more based on fast and incremental deployments with multiple development cycles. The same consideration also holds for adaptive Web applications and their adaptivity requirements. Our approach allows us to abstract the adaptive behaviors, to extract them from the main application logic, and to provide a decoupled management support, finally enhancing the maintainability and evolvability of the overall application.

7.4 Conclusion and Future Works

In this chapter we described two different areas where the results from Chapter 4 and Chapter 5 have been applied successfully. We then identified a limitation that in general characterizes conceptual modeling approaches, i.e. the focus on design time support, and we outlined a possible solution to this shortcoming.

While the MAIS project was concluded in June 2006, the research on WBM (Section 7.2) for behavior-aware Web applications and the research on ECA rules for the runtime management of adaptivity (Section 7.3) is still ongoing and leaves room for improvements:

- In our future work on WBM, we plan to extend our initial prototype to support more sophisticated policies for dealing with priorities and conflicts. Currently, we adopt the simple policy of always choosing the rule of highest priority for execution. Furthermore, we did not yet consider the problems of rule termination, which might arise when rules trigger each other. Thanks to the strict relation between WBM and WebML we believe that some termination problem can be detected at design time, but it is neither possible nor useful to constrain rule sets at design time so as to avoid any cyclic behavior. Therefore, we will need to support monitoring of nonterminating behaviors at runtime. Also, the dynamic

activation and deactivation of rules and of rule groups is already under consideration, and the current prototype includes a preliminary version of such features. Our first prototype demonstrates the applicability of our approach, as it supports complex rules and enables the design of complex, behavior-aware applications.

- In our future work ECA-Web we will focus on the extension of the ECA-Web language to fully take advantage of the concepts and notations that can be extracted from conceptual Web application models (i.e. in our case, from WebML models). We will also investigate termination, complexity, and confluence issues, trying to apply Chimera-Exception's Termination Analysis Machine [91] to ECA-Web. Extensive experimentations are planned to further prove the advantages deriving from the decoupled approach.

8 Conclusion

In this dissertation we have considered two relevant aspects of modern Web applications, i.e. context-awareness and adaptivity, and we have shown how such aspects require increasing the expressive power of Web application models so as to incorporate changes in the page generation logic that support adaptation. The proposed approach, based on WebML, enables the automatic generation of adaptive Web applications, starting from properly extended WebML models and supported by a suitably adapted runtime environment.

The solutions described in this dissertation have been extensively tested in the context of the Italian research project MAIS by altering the runtime component of the WebRatio tool and by developing a context-aware prototype application. These activities allowed us to prove that the proposed solution is feasible and meets an important customer demand.

8.1 Results and Contributions

The *results* that have been achieved in the context of the research described in this dissertation can be summarized as follows:

- *Concepts and techniques* from the fields of context-aware computing, context modeling, ubiquitous and/or pervasive computing have been applied successfully to the domain of the Web.
- The novel ideas have been appropriately formalized by extending a well-known conceptual *modeling language/method* for the design of Web applications, i.e. WebML [83, 88, 92], keeping the intuitive, visual modeling paradigm that characterizes the language.
- The extended modeling method has yielded the realization of a proper *modeling instrument* by adapting the visual CASE tool for WebML. The tool is equipped with a powerful automatic code generator that enables the fast and efficient implementation of adaptive Web applications [88]. The strong compliance with WebML and WebRatio maximizes the extensibility of the developed solution.

- The proposed modeling method has been adopted as *conceptual basis* for a set of correlated research activities:
  - The method has been adopted in the MAIS project and has been the basis for the activities of other involved partners. The joint work led to the development of two prototype applications [70, 72, 93], which allowed us to perform further *experimentations*.
  - The instrument and its extensibility have been leveraged for the *experimentation* of behavior-aware Web applications to enable the use of composite navigation events. Composite events are expressed by means of WBM (Web Behavior Model) and trigger adaptivity [94, 84, 85, 95].
  - The experimentations have yielded the idea of a further *extension* of the proposed solution to also support the runtime management of adaptivity. For this purpose, the adoption of a decoupled rule engine has been proposed [90, 96].

Table 8.1 summarizes the features of the proposed model according to the dimensions used to compare the level of adaptivity support present in other conceptual design methods. With respect to such dimensions, we can state that our method covers the most relevant features.

Starting from these results, we can summarize the *contributions* to the Web Engineering research field contained in this dissertation. We have shown the feasibility and viability of context-aware solutions also in a "non-traditional" field, such as the one represented by Web applications. In fact, till now context-awareness has mainly been studied in the fields of ubiquitous or pervasive computing, and typical mobile applications (e.g. GPS navigator systems) have been primarily based on proprietary client-side solutions. In this regard, the presented approach is one of the first attempts to enlarge the applicability of adaptive application features in the Web from "adaptive hypermedia systems" to "context-aware Web applications". While the former typically are based on dynamically updated user profiles based on the user's browsing activities, the latter may be based on more complex context models and active, context-triggered application features, e.g. supporting mobile applications.

We have provided a conceptualization of the issues that may arise when introducing context-awareness to the discipline of Web engineering, and, along with this conceptualization, we have extended the terminology used to describe adaptive or context-aware Web applications. The terminology used in similar modeling approaches shows that there

Method	Dimension	Evaluation
WebML	Adaptability	This dissertation shows how adaptability to user preferences and device characteristics is supported by WebML.
	Adaptivity	This dissertation introduces support for run-time adaptation into the conceptual modeling method.
	Context-Awareness	This dissertation further shows how context can be modeled and promotes active context-awareness.
	Modeling Paradigm	This dissertation introduces context-awareness and adaptivity in a WebML-consistent visual fashion.
	Tool Support	This dissertation provides an extension of the WebRatio CASE tool, that allows the fully visual design and the automatic generation of the application code.

Table 8.1: Positioning of the extended WebML design method with respect to the comparison dimensions introduced to compare the support for adaptation of other conceptual modeling methods (see Section 2.4 and Table 2.1 for the comparison).

are still subtle differences and, also, in the solutions proposed. We hope that this dissertation also contributes to make a small step forward toward a common terminological framework in the area of adaptive and/or context-aware Web applications.

Although our conceptualization and terminology are applied to the WebML development method, the proposed design primitives and ideas are general in nature, and may thus be easily applied to other Web design methods as well.

8.2 Limitations

Although we have tried to cover the most salient aspects of adaptivity in the Web, our approach has some limitations:

- We have not yet considered *adaptive link hiding or showing*, based on the state of context or user profile data, which is an interesting and characterizing feature of adaptive Web applications.

- Due to our initial assumption about the use of traditional Web technologies and the classical three-tier architecture, the *smallest*

object of adaptation in this dissertation is the page, i.e. adaptivity is possible only by re-computing an entire page.

- The *tool support* for the visual modeling solution described in this dissertation could only be partially implemented as extension of the WebRatio CASE tool, i.e. the hierarchical distribution of adaptivity actions by means of adaptive/context-aware containers could not be supported. However, this does not limit the actual adaptivity features supported, as actions that would have been specified at container level can equally be specified at page level.

- We have not yet been able to perform an exhaustive *usability study* with big numbers of users, so as to check the interweaving of user- and context-driven adaptations.

- Despite the benefits of the *visual modeling paradigm*, the specification of large sets of adaptivity actions attached to single pages may cause an overloading of the otherwise intuitive visual hypertext schemas.

- The decision to focus on the delivery of context-aware features to thin clients requires us to maintain the context model at *server side*, where also the necessary adaptation logic resides (i.e. specified in the adaptive hypertext). This prevents the current solution from delegating, for example, context data transformations to the client.

8.3 Ongoing and Future Work

As a continuation of this research, we are planning to address some of the restrictions described in the previous section and a few new issues that we believe are of particular interest for context-awareness and adaptivity in Web applications. In particular, our planned ongoing and future work addresses the first four of the described limitations and also focuses on a few new aspects:

- We will investigate mechanisms for *adaptive link hiding and showing* and, more in general, for adaptation of presentation properties. For this purpose we will study the potential of post-processing mechanisms.

- We are planning to investigate how the adoption of technologies for *Rich Internet Applications* (RIAs [97, 98]) may enable adaptations

to address a *finer granularity*, shifting adaptive behaviors from entire pages to single page components, such as individual content units or links.

- Concerning the current *implementation*, we will try to further extend the WebRatio CASE tool, so as to fully support the modeling paradigm introduced in this dissertation and to provide for the complete automatic code generation of adaptive Web applications. As such, the extension could be integrated in a future release of the WebRatio environment.

- In order to verify the *usability* of the proposed adaptivity paradigm, we are planning experiments involving larger numbers of real users. The aim of this study is to investigate the right balance between user-controlled interactions and context-triggered adaptations.

- We are planning to *formally model* some aspects of the introduced rule modeling logic to analyze possible conflicts that may be caused by the concurrent activation of adaptivity actions.

- We are studying adaptivity modes that 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 an extension of the client-side logic).

- We will study how *Web services* and the SOAP protocol can be adopted to enhance the background context monitoring solution outlined in Section 5.3.

- Going one step further, in a framework where Web applications result from the *composition* of other stand-alone applications, as envisioned in [99] and [100], a prominent research challenge is to understand how adaptability and adaptivity may impact on single components as well as on the resulting composite application.

Bibliography

[1] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera, *Designing Data-Intensive Web Applications*, Morgan Kauffmann, 2002.

[2] Web Models s.r.l., "WebRatio Site Development Studio," http://www.webratio.com, 2005.

[3] Barry Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[4] Kent Beck, "Embracing Change with Extreme Programming," *IEEE Computer*, vol. 32, no. 10, 1999.

[5] Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*, Object Technology Series. Addison-Wesley Professional, 1999.

[6] Jim Conallen, *Building Web applications with UML*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[7] Anind K. Dey and Gregory D. Abowd, "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, 2000.

[8] Flavius Frasincar and Geert-Jan Houben, "Hypermedia Presentation Adaptation on the Semantic Web," in *Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'02)*, Málaga, Spain, 2002, pp. 133–142, Springer-Verlag.

[9] Mark Weiser, "The Computer for the 21st Century," *Scientific American*, September 1991.

[10] Bill N. Schilit and Marvin M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *IEEE Network*, vol. 8, no. 5, pp. 22–32, September/October 1994.

[11] Anind K. Dey and Gregory D. Abowd, "Towards a Better Understanding of Context and Context-Awareness," Technical Report GIT-GVU-99-22, Georgia Institute of Technology, 1999.

[12] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons, "The Active Badge Location System," *ACM Transactions on Information Systems*, vol. 10, no. 1, pp. 91–102, 1992.

[13] Gregory D. Abowd, Christopher G. Atkeson, Jason I. Hong, Sue Long, Rob Kooper, and Mike Pinkerton, "Cyberguide: A mobile context-aware tour guide," *Wireless Networks*, vol. 3, no. 5, pp. 421–433, 1997.

[14] Gerti Kappel, Birgit Proll, Werner Retschitzegger, and Wieland Schwinger, "Customization for Ubiquitous Web Applications – A Comparison of Approaches," *International Journal of Web Engineering and Technology*, January 2003.

[15] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde, "Advanced Interaction in Context," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing (HUC'99)*, London, UK, 1999, pp. 89–101, Springer-Verlag.

[16] Daniel Salber, Anind K. Dey, and Gregory D. Abowd, "The Context Toolkit: Aiding the Development of Context-Enabled Applications," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1999, pp. 434–441, ACM Press.

[17] Thomas Strang and Claudia Linnhoff-Popien, "A Context-Modeling Survey," in *Workshop on Advanced Context Modelling, Reasoning and Management (part of UbiComp'04)*, September 2004.

[18] Graham Klyne, Franklin Reynolds, Chris Woodrow, Hidetaka Ohto, Johan Hjelm, Mark H. Butler, and Luu Tran, "Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0," W3C Recommendation, W3C, January 2004.

[19] Open Mobile Alliance Ltd., "User Agent Profile (UAProf)," http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html, 2006.

[20] Albert Held, Sven Buchholz, and Alexander Schill, "Modeling of Context Information for Pervasive Computing Applications," in

Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI'02), Orlando, Florida, July 2002.

[21] Roberto De Virgilio and Riccardo Torlone, "Modeling heterogeneous context information in adaptive web based applications," in *Proceedings of the 6th international conference on Web engineering (ICWE'06)*. 2006, pp. 56–63, ACM Press.

[22] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy, "Generating context management infrastructure from context models," in *4th International Conference on Mobile Data Management (MDM'03), Industrial Track Proceedings*, Melbourne, Australia, January 2003, pp. 1–6.

[23] Terry Halpin, *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*, Morgan Kaufmann, 2001.

[24] Keith Cheverst, Keith Mitchell, and Nigel Davies, "Design of an Object Model for a Context Sensitive Tourist GUIDE," *Computers and Graphics*, vol. 23, no. 6, pp. 883–891, 1999.

[25] John McCarthy, "Notes on formalizing context," in *Proceeding of the 13th International Joint Conference on Artificial Intelligence*, August 1993, pp. 555–560.

[26] Varol Akman and Mehmet Surav, "The Use of Situation Theory in Context Modeling," *Computational Intelligence*, vol. 12, no. 4, 1996.

[27] Harry Chen, Tim Finin, and Anupam Joshi, "Using OWL in a Pervasive Computing Broker," in *Proceedings of the Workshop on Ontologies in Open Agent Systems (AAMAS'03)*, 2003.

[28] Peter Brusilovsky, "Methods and Techniques of Adaptive Hypermedia," *User Modeling and User-Adapted Interaction*, vol. 6, no. 2-3, pp. 87–129, 1996.

[29] Sue Long, Rob Kooper, Gregory D. Abowd, and Christopher G. Atkeson, "Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study," in *MOBICOM*, 1996, pp. 97–107.

[30] Frank Allan Hansen, Niels Olof Bouvin, Bent G. Christensen, Kaj Grønbæk, Torben Bach Pedersen, and Jevgenij Gagach, "Integrat-

ing the Web and the world: contextual trails on the move," in *Hypertext*, 2004, pp. 98–107.

[31] Paul De Bra, Ad Aerts, Bart Berden, Barend de Lange, Brendan Rousseau, Tomi Santic, David Smits, and Natalia Stash, "AHA! The adaptive hypermedia architecture," in *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia (HYPERTEXT'03)*, 2003, pp. 81–84.

[32] Rudi Belotti, Corsin Decurtins, Michael Grossniklaus, Moira C. Norrie, and Alexios Palinginis, "Interplay of Content and Context," in *Proceedings of ICWE'04*, 2004, pp. 187–200.

[33] Michael Grossniklaus and Moira C. Norrie, "Information Concepts for Content Management," in *WISE Workshops*, 2002, pp. 150–159.

[34] Richard Vdovjak, Flavius Frasincar, Geert-Jan Houben, and Peter Barna, "Engineering Semantic Web Information Systems in Hera," *Journal of Web Engineering*, vol. 2, no. 1-2, pp. 3–26, 2003.

[35] Daniel Schwabe, Gustavo Rossi, and Simone D. J. Barbosa, "Systematic hypermedia application design with OOHDM," in *Proceedings of the the seventh ACM conference on Hypertext (HYPERTEXT'96)*, New York, NY, USA, 1996, pp. 116–128, ACM Press.

[36] Daniel Schwabe and Gustavo Rossi, "An object oriented approach to Web-based applications design," *Theory and Practice of Object Systems*, vol. 4, no. 4, pp. 207–225, 1998.

[37] Jaime Gómez, Cristina Cachero, and Oscar Pastor, "Extending a Conceptual Modelling Approach to Web Application Design," in *Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE'00)*, London, UK, 2000, pp. 79–93, Springer-Verlag.

[38] Olga De Troyer and Tom Decruyenaere, "Conceptual modelling of web sites for end-users," *World Wide Web Journal*, vol. 3, no. 1, pp. 27–42, 2000.

[39] Nora Koch, Andreas Kraus, and Rolf Hennicker, "The Authoring Process of the UML-based Web Engineering Approach," in *First International Workshop on Web-oriented Software Technology (IWWOST01)*, Daniel Schwabe, Ed., 2001.

[40] Yuhui Jin, Stefan Decker, and Gio Wiederhold, "OntoWebber: Model-Driven Ontology-Based Web Site Management," in *The 1st International Semantic Web Working Symposium (SWWS'01), Stanford University, Stanford, CA, July 29-Aug 1.* 2001, Springer Verlag.

[41] Bernhard Thalheim and Antje Düsterhöft, "SiteLang: Conceptual Modeling of Internet Sites," in *Proceedings of the 20th International Conference on Conceptual Modeling (ER'01)*, London, UK, 2001, pp. 179–192, Springer-Verlag.

[42] Flavius Frasincar, Peter Barna, Geert-Jan Houben, and Zoltán Fiala, "Adaptation and Reuse in Designing Web Information Systems," in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2*, Washington, DC, USA, 2004, IEEE Computer Society.

[43] Daniel Schwabe, Robson Guimaraes, and Gustavo Rossi, "Cohesive Design of Personalized Web Applications," *IEEE Internet Computing*, vol. 6, no. 2, pp. 34–43, March-April 2002.

[44] Irene Garrigós, Sven Casteleyn, and Jaime Gómez, "A Structured Approach to Personalize Websites Using the OO-H Personalization Framework," in *Web Technologies Research and Development - APWeb 2005.* 2005, vol. 3399/2005 of *LNCS*, pp. 695–706, Springer Berlin / Heidelberg.

[45] Sven Casteleyn, Olga De Troyer, and Saar Brockmans, "Design time support for adaptive behavior in Web sites," in *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC'03)*, New York, NY, USA, 2003, pp. 1222–1228, ACM Press.

[46] Hubert Baumeister, Alexander Knapp, Nora Koch, and Gefei Zang, "Modeling Adaptivity with Aspects," in *Proceedings of ICWE 2005, Sydney, Australia.*, D. Lowe and M. Gaedke, Eds. July 2005, vol. 3579 of *LNCS*, pp. 406–416, Springer-Verlag Berlin Heidelberg.

[47] Robert E. Filman, Tzilla Elrad, Siobhan Clarke, and Mehmet Aksit, *Aspect-Oriented Software Development*, Addison-Wesley, 2004.

[48] Klaus-Dieter Schewe and Bernhard Thalheim, "Reasoning about web information systems using story algebras," in *Advances in Databases and Information Systems (ADBIS'04), volume 3255 of*

Lecture Notes in Computer Science. 2004, pp. 54–66, Springer Verlag.

[49] Dexter Kozen, "Kleene algebra with tests," *ACM Transactions on Programming Languages and Systems*, vol. 19, no. 3, pp. 427–443, 1997.

[50] Aleksander Binemann-Zdanowicz, Roland Kaschek, Klaus-Dieter Schewe, and Bernhard Thalheim, "Context-aware Web Information Systems," in *Proceedings of the first Asian-Pacific conference on Conceptual modelling (APCCM'04)*, Darlinghurst, Australia, Australia, 2004, pp. 37–48, Australian Computer Society, Inc.

[51] Flavius Frasincar, Geert-Jan Houben, and Peter Barna, "Hera presentation generator," in *Special interest tracks and posters of the 14th international conference on World Wide Web (WWW'05)*, New York, NY, USA, 2005, pp. 952–953, ACM Press.

[52] Daniel Schwabe, Rita de Almeida Pontes, and Isbela Moura, "OOHDM-Web: an environment for implementation of hypermedia applications in the WWW," *ACM SIGWEB Newsletter*, vol. 8, no. 2, pp. 18–34, 1999.

[53] Jaime Gómez, "Model-Driven Web Development with VisualWADE," in *Proceeding of the 4th International Conference on Web Engineering (ICWE'04)*, 2004, pp. 611–612.

[54] Yuhui Jin, Sichun Xu, Stefan Decker, and Gio Wiederhold, "Managing Web Sites with OntoWebber," in *Proceedings of the 8th International Conference on Extending Database Technology (EDBT'02)*, London, UK, 2002, pp. 766–768, Springer-Verlag.

[55] Bernhard Thalheim, Klaus-Dieter Schewe, Irina Romalis, Thomas Raak, and Gunar Fiedler, "Website Modeling and Website Generation," in *4th International Conference on Web Engineering (ICWE'04)*. 2004, vol. 3140 of *LNCS*, pp. 577–578, Springer Berlin/Heidelberg.

[56] Franca Garzotto, Paolo Paolini, and Daniel Schwabe, "HDM – a model-based approach to hypertext application design," *ACM Transactions on Information Systems*, vol. 11, no. 1, pp. 1–26, 1993.

[57] Tomás Isakowitz, Edward A. Stohr, and P. Balasubramanian, "RMM: a methodology for structured hypermedia design," *Communications of the ACM*, vol. 38, no. 8, pp. 34–44, 1995.

[58] Ioana Manolescu, Marco Brambilla, Stefano Ceri, Sara Comai, and Piero Fraternali, "Model-driven design and deployment of service-enabled Web applications," *ACM Transactions on Internet Technology*, vol. 5, no. 3, pp. 439–479, 2005.

[59] Marco Brambilla, Stefano Ceri, Piero Fraternali, and Ioana Manolescu, "Process Modeling in Web Applications," *ACM Transactions on Software Engineering and Methodology*, 2006.

[60] Karen Henricksen and Jadwiga Indulska, "Modelling and Using Imperfect Context Information," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW'04)*, Washington, DC, USA, 2004, pp. 33–37, IEEE Computer Society.

[61] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy, "Modeling Context Information in Pervasive Computing Systems," in *Proceedings of the First International Conference on Pervasive Computing (Pervasive'02)*, London, UK, 2002, pp. 167–180, Springer-Verlag.

[62] Hui Lei, Daby M. Sow, John S. Davis, II, Guruduth Banavar, and Maria R. Ebling, "The design and applications of a context service," *SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 45–55, 2002.

[63] Paul De Bra, Geert-Jan Houben, and Hongjing Wu, "AHAM: a Dexter-based reference model for adaptive hypermedia," in *Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots (HYPERTEXT'99)*, 1999, pp. 147–156.

[64] Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali, and Ioana Manolescu, "Specification and Design of Workflow-Driven Hypertexts," *Journal of Web Engineering*, vol. 1, no. 2, pp. 1–100, April 2003.

[65] Jennifer Widom and Stefano Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing*, Morgan Kaufmann Publishers, 1996.

[66] Alexander Aiken, Jennifer Widom, and Joseph M. Hellerstein, "Behavior of database production rules: termination, confluence, and

observable determinism," in *Proceedings of the 1992 ACM SIGMOD international conference on Management of data (SIGMOD '92)*, New York, NY, USA, 1992, pp. 59–68, ACM Press.

[67] Elena Baralis and Jennifer Widom, "An Algebraic Approach to Rule Analysis in Expert Database Systems," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, San Francisco, CA, USA, 1994, pp. 475–486, Morgan Kaufmann Publishers Inc.

[68] Barbara Pernici (Ed.), *Mobile Information Systems - Infrastructure and Design for Adaptivity and Flexibility*, Springer Verlag, April 2006.

[69] MAIS Consortium, "MAIS Project Home Page," http://www.mais-project.it, October 2006.

[70] Maurizio Brioschi, Stefano Ceri, Florian Daniel, Federico M. Facca, Gabriele Giunta, Maristella Matera, Domenico Presenza, and Marco Riva, "Primo prototipo di uno strumento per la produzione di siti Web multicanali personalizzati," Prototype P. 7.1.1, The MAIS Consortium, July 2004.

[71] Malcolm Davis, "Struts, an open-source MVC implementation," February 2001, http://www-106.ibm.com/developerworks/library/j-struts/?n-j-2151.

[72] Maurizio Brioschi, Stefano Ceri, Florian Daniel, Federico M. Facca, Massimo Legnani, and Maristella Matera, "Secondo prototipo di uno strumento per la produzione di siti web multicanale personalizzati," Prototype P. 7.1.2, The MAIS Consortium, September 2005.

[73] The WebRatio team, "Custom Units Tutorial and Reference Guide (WebRatio 4.2)," Tech. Rep., Web Models s.r.l., March 2006.

[74] The WebRatio team, "EasyStyle User and Reference Guide (WebRatio 4.2)," Tech. Rep., Web Models s.r.l., March 2006.

[75] Chaeron Corporation, "Chaeron GPS (Global Positioning System) Library," http://www.chaeron.com/gps.html, 2005.

[76] Place Lab, "Place Lab: A privacy-observant location system," http://www.placelab.org, 2006.

[77] Florian Daniel and Maristella Matera, "Rapporto conclusivo," Technical report, MAIS Consortium, R.7.1.4 2005.

[78] WWW Consortium, "Extensible stylesheet language (xsl) version 1.1," W3C Recommendation, http://www.w3.org/TR/2006/REC-xsl11-20061205/, 2006.

[79] Sandia National Laboratories, "Jess - the Rule Engine for the Java Platform," http://herzberg.ca.sandia.gov/jess/index.shtml, 2005.

[80] W3C, "W3C Recommendation - Modularization of XHTML," http://www.w3.org/TR/xhtml-modularization/, 2001.

[81] Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali, and Ioana Manolescu, "Specification and Design of Workflow-Driven Hypertexts," *Journal of Web Engineering (JWE)*, vol. 1, no. 2, pp. 163–182, April 2003.

[82] Federico Michele Facca and Pier Luca Lanzi, "Mining interesting knowledge from weblogs: a survey," *Data Knowledge Engineering*, vol. 53, no. 3, pp. 225–241, 2005.

[83] Stefano Ceri, Florian Daniel, and Maristella Matera, "Extending WebML for Modeling Multi-Channel Context-Aware Web Applications," in *Proceedings of Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03), Rome, Italy, December 12 -13, 2003.* 2003, pp. 225–233, IEEE Press.

[84] Stefano Ceri, Florian Daniel, Vera Demaldé, and Federico Michele Facca, "An Approach to User-Behavior-Aware Web Applications," in *Proceedings of the 5th International Conference on Web Engineering (ICWE'05), Sydney, Australia, July 27-29.* 2005, vol. 3579 of *LNCS*, pp. 417–428, Springer Verlag.

[85] Stefano Ceri, Florian Daniel, and Federico M. Facca, "Modeling Web Applications reacting to User Behaviors," *Elsevier Computer Networks*, vol. 50, no. 10, pp. 1533–1546, July 2006.

[86] Rajeev Alur and David L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[87] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine, "Symbolic model checking for real-time systems," *Information and Computation*, vol. 111, no. 2, pp. 193–244, 1994.

[88] Stefano Ceri, Florian Daniel, Maristella Matera, and Federico M. Facca, "Model-driven Development of Context-Aware Web Applications," *ACM Transactions on Internet Technology*, vol. 7, no. 1, February 2007.

[89] Irene Garrigós, Jaime Gómez, Peter Barna, and Geert-Jan Houben, "A reusable personalization model in web application design," in *Proceedings of The ICWE 2005 Workshop on Web Information Systems Modelling (WISM'05), Sydney, Australia.* 2005, pp. 40–49, University of Wollongong, School of IT and Computer Science.

[90] Florian Daniel, Maristella Matera, and Giuseppe Pozzi, "Combining Conceptual Modeling and Active Rules for the Design of Adaptive Web Applications," in *Workshop proceedings of the sixth international conference on Web Engineering (ICWE'06)*, New York, NY, USA, 2006, p. 10, ACM Press.

[91] Fabio Casati, Stefano Ceri, Stefano Paraboschi, and Guiseppe Pozzi, "Specification and implementation of exceptions in workflow management systems," *ACM Transactions on Database Systems*, vol. 24, no. 3, pp. 405–451, 1999.

[92] Stefano Ceri, Florian Daniel, Federico M. Facca, and Maristella Matera, "Model-Driven Engineering of Active Context-Awareness," *World Wide Web Journal*, 2007, (In print).

[93] Riccardo Torlone et al., "Methods and Tools for the Development of Adaptive Applications," in *Mobile Information Systems. Infrastructure and Design for Flexibility and Adaptivity*, Barbara Pernici, Ed., pp. 209–247. Springer Verlag, April 2006.

[94] Federico Michele Facca, Stefano Ceri, Jacopo Armani, and Vera Demaldé, "Building Reactive Web Applications," in *Proceedings of the 14th international conference on World Wide Web (WWW'05), Chiba, Japan, May 10-14, 2005 - Special interest tracks and posters.* 2005, pp. 1058–1059, ACM.

[95] Federico M. Facca and Florian Daniel, "Visual Modeling of ReActive Web Applications," in *Current Trends in Database Technology - EDBT 2006.* 2006, vol. 4254 of *LNCS*, pp. 876–886, Springer Berlin/Heidelberg.

[96] Florian Daniel, Maristella Matera, Alessandro Morandi, Matteo Mortari, and Giuseppe Pozzi, "Active Rules for Runtime Adaptivity Management," in *Proceedings of the Seventh International*

Conference on Web Engineering (ICWEŠ07), Como, Italy, July 2007, LNCS, Springer.

[97] Macromedia Inc., "Developing Rich Internet Applications with Macromedia MX 2004," Macromedia White Paper, August 2003.

[98] Laszlo Systems Inc., "OpenLaszlo - An XML Framework for Rich Internet Applications," Laszlo Systems Technology White Paper, July 2006.

[99] Florian Daniel, Maristella Matera, Jin Yu, Boualem Benatallah, Regis Saint-Paul, and Fabio Casati, "Understanding UI Integration: A survey of problems, technologies," *IEEE Internet Computing*, May-June 2007, (In print).

[100] Jin Yu, Boualem Benatallah, Regis Saint-Paul, Fabio Casati, Florian Daniel, and Maristella Matera, "A Framework for Rapid Integration of Presentation Components," in *Proceedings of the 16th International World Wide Web Conference*, Banff, Alberta, Canada, May 2007, ACM Press, (In print).

Index