Chapter 3

Development of Adaptive Web Applications: State of the Art

"If you wish your merit to be known, acknowledge that of other people."[1]

The previous chapter gave a short introduction to adaptive hypermedia and Web-based systems. Basic definitions were stated, and reference models for adaptive hypermedia applications were presented. Furthermore, the most important methods, techniques, and application areas of hypermedia and Web adaptation were summarized.

The continually increasing complexity of Web sites, the heterogeneity of their audience, and the growing diversity of available Web client devices make adaptation (to the user, his device, and entire usage context) to a crucial issue of today's Web-based systems. Nevertheless, this need for adaptation leads to additional requirements towards the anyhow complex development process of Web applications. As stated in Chapter 1, these additional requirements concern all phases of a Web application's life cycle: design, implementation, publication, testing, maintenance, etc. Hence, it becomes obvious that the development of adaptive Web-based systems needs to be based on systematic engineering approaches that allow to take into account personalization and device independence in a structured manner.

Whereas the development of early Web sites was characterized by ad hoc approaches, in the last decade new initiatives have been undertaken to address the complexity and problems involved in creating and maintaining Web-based systems. Since about 2000, one can talk about a new *Web engineering* discipline. The term Web engineering itself is defined by Murugesan et al. in [Murugesan et al. 2001] as follows:

Definition 3.1 (Web Engineering) *"Web engineering is the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems and applications."*

Web engineering is a multidisciplinary field that encompasses inputs from diverse areas such as software engineering, human-computer interaction, user interface design, requirements engineering, systems analysis and design, hypermedia or multimedia [Deshpande et al. 2002]. It addresses numerous aspects of Web application development and management, among them design methods and methodologies; implementation techniques; usability issues; testing, verification and validation; performance specification and analysis; update and maintenance etc [Murugesan and Deshpande 2001]. Furthermore, triggered by the above mentioned need

[1] Oriental proverb

for personalization and device/context dependency, an increasing amount of research is devoted to *adaptation engineering*, as well.

The goal of this chapter is to provide an overview of (a well-defined subset of) existing Web engineering approaches aimed at the development of adaptive Web applications. It is basically divided into two main parts: one (Section 3.2) summarizing component-based and document-centric Web engineering approaches, the other (Section 3.3) focusing on model-based Web or hypermedia design methods and methodologies. Yet, to appropriately situate these topics (and thus the main focus of this dissertation) in the overall Web engineering research field, Section 3.1 provides a short overview of the typical life-cycle of (adaptive) Web applications.

3.1 Overview of the Overall Web Engineering Life-Cycle

As stated above, the development and maintenance of complex Web applications should be based on systematic engineering processes and principles. Most typically, one can distinguish between following process phases and related Web engineering activities [Kappel et al. 2004].

Requirements Analysis: Similar to traditional software engineering, the development of a Web application typically starts with a requirements analysis. The corresponding *requirements engineering* (RE) discipline covers all activities related to the ascertainment, documentation, validation, and maintenance of requirements [Grünbacher 2003]; and involves all stakeholders (e.g. the providers, developers, but also the targeted users) of the planned Web site. The gathered requirements might be both functional or non-functional and address the application itself (e.g. supported user groups, the provided content, or functionality), its systems environment (hardware or software components), or even the entire project plan (deliverables, budget, etc.). They can be described in different ways, such as in form of stories, storyboards, requirements lists, use cases, etc.

In contrast to conventional software systems, the requirements engineering for Web sites is characterized by multidisciplinarity (i.e. the participation of different domain experts like multimedia experts, content authors, database specialists, etc.), a larger importance of quality factors (performance, security, usability, accessibility), and the need to consider heterogeneous and dynamically changing deployment environments [Lowe 2003]. In the case of adaptive Web sites, this heterogeneity even increases since analysts have to consider very different end devices, user groups, and usage contexts.

Design and Modeling: The requirements analysis phase is followed by the design of the Web site, mostly based on a Web design method [Schwinger and Koch 2003]. Addressing different dimensions of the problem area, such methods allow specifying hypermedia applications in an appropriate level of abstraction. Even though not (yet) widespread in practice, the Web engineering research field has recently proposed a number of *model-based Web design methods*[2], the most important of which will be described in detail in Section 3.3. As will be shown, they typically distinguish the data, the hypertext, and the user interface aspects of a Web-based system.

[2]Note that throughout this dissertation (but also in general in the Web and software engineering fields), the term "model" refers to different concepts, e.g. to reference models, component models, user models, context models, design models, etc. To avoid confusion, the author aims to use the term by always explicitly naming its particular context, as far as this is not unambiguously clear from the actual section or paragraph.

When designing personalized ubiquitous Web applications, designers have to deal with the additional aspect of *contextuality*. This requires a modeling of both different adaptation targets (e.g. users, client devices, usage contexts) and the appropriate adaptation processes in a high level of abstraction. However, the fact that hypermedia adaptation can concern all aspects of a Web site (content, navigation, presentation (see Section 2.2.3)) implies a thorough reconsideration of (all models of) a "non-adaptive" Web design.

Implementation: After specifying (designing) "what" functionality a Web-based system should provide, the next question is "how" this should be implemented. Due to the distributed nature and the recent rapid evolution of the WWW, there exists meanwhile a huge number of different implementation techniques, among them document-centric markup languages (e.g. HTML, WML, or other XML-based grammars), client-side scripting and programming tools (plug-in technologies, applets, Java- or ActiveScript, etc.), as well as server-side technologies (code-generators, database systems, application frameworks)[Gaedke et al. 2003]. The realization of a complex Web application requires not the application of one single technology, rather the combination of different technologies within a comprehensive implementation framework. Furthermore, to efficiently cope with complexity, there is a crucial need for implementation techniques providing for "black-box-like" reuse and configurability of Web code artefacs both in different implementation phases and on varying levels of granularity.

These aspects of reusability and configurability become even more important when realizing adaptive Web applications. Instead of separately creating and managing content and/or Web code for different client environments or possible user groups, there is a need for "write-once-publish-everywhere" technologies enabling to create adaptable (or even self-adaptive) implementation artefacts that can be (automatically) adjusted to different application contexts. Thus, implementation technologies are required that provide a clear separation of concerns (like content, layout, navigation, interaction behavior, etc.) in a reusable manner.

Test: The testing of a Web application has a crucial role in the overall Web site process [Lam 2001]. It addresses both the functional and qualitative behavior of an application, i.e. "a Web-based system needs to be tested not only to check and verify whether it does what it is designed to do but also to evaluate how well it performs in (different) Web client environments [Murugesan et al. 2001]". Steindl et al. [Steindl et al. 2003] distinguish between different dimensions of Web testing: 1) the dimension of quality characteristics (e.g. usability, security, performance) to be tested, 2) the dimension of system components (links, pages, server infrastructure components) to be investigated, and 3) the phases of the overall Web engineering process (e.g. implementation, system integration, deployment) that require test efforts. Meanwhile, there exists a broad repertoire of test techniques and tools, ranging from link checker tools to complex stress test frameworks for multi-tier Web architectures.

The testing of adaptive Web applications implies additional challenges compared to non-adaptive Web sites. The first problem to be considered is the fact that their behavior can significantly vary depending on the actual usage context. However, with a growing number of context variables, the investigation of each test case for every possible context configuration becomes soon unmanageable. Furthermore, it is also nearly impossible to foresee all conceivable context configurations, e.g. the developers of a device-independent Web site can not test it for every existing Web-capable end

device. A second problem is that the continuous acquisition and modeling of user, usage, and context information causes additional server load, i.e. it can easily lead to increasing response times and a worse system performance. Consequently, there is a need for Web testing approaches that explicitly consider these additional requirements.

Operation and Maintenance: Once created, deployed, and tested, a Web application can be launched and made accessible for its targeted audience. Yet, in contrast to traditional software applications, Web sites require continuous maintenance and updates even in their operational phase [Deshpande et al. 2002]. Web maintenance comprises a number of different activities, such as the advertisement of a Web application, the steady observation and fine-tuning of its configuration (Web Configuration Management [Dart 1999]), but first of all the continuous updating/refreshing of its content. This latter aspect of Web Content Management (WCM [Fiala 2001]) is especially crucial, since Web sites are in first place information systems, i.e. their popularity strongly depends on the quality and actuality of the published content.

Like on all other engineering phases, the consideration of adaptation has additional implications on Web maintenance. First, content creators (e.g. graphics designers, news editors) have to prepare different versions of their content assets so that it optimally fits the requirements of different user groups and client devices. Second, the evolutionary nature of the Web also requires Web providers to continually "readjust or reconfigure" even a running Web system to meet the characteristics of a newly emerged client device or a previously not considered usage context. Yet, apart from a few publications (e.g. [Belotti et al. 2005]), this interplay of context-dependency and Web maintenance has not been sufficiently addressed by academia, yet.

As can be seen, adaptation and personalization have a significant impact on the overall life-cycle of a Web application. However, as a matter of course, this dissertation can only deal with a small subset of the whole Web engineering process in detail. As stated in Chapter 1, the central topic is the model-based design and component-based implementation of adaptive Web sites. Therefore, this chapter consists of two parts: one reviewing component-based and document-oriented Web engineering solutions, the other surveying model-based Web design methods.

The component- and document-oriented approaches discussed in Section 3.2 are basically centered around the implementation and presentation aspects of Web applications. Considering the document-centric nature of the Web (or in more general of hypertext and hypermedia systems), they provide formalized (mostly XML-based) languages for efficiently creating and publishing Web presentations. Hence abstracting from the current coarse-grained implementation model of the Web, they facilitate a number of benefits, such as reusability, configurability, personalization, as well as platform and device independence. The documents created in these languages typically represent Web implementation artefacts on different abstraction levels (i.e. from atomic resources to complex Web presentations) and can be automatically transformed to a specific Web or multimedia implementation format (e.g. (X)HTML, WML, SMIL, X3D, etc.).

On the other hand, the model-based Web design methods described in Section 3.3 pursue a complementary goal. Applying well-approved concepts, methods, and techniques of the Model-Driven Software Engineering (MDE) paradigm to the particularities of Web-based applications, they provide structured conceptual design methodologies that clearly separate the different design issues involved in the design of Web applications [Costagliola et al. 2002]. All these different design issues are dealt with in separate design steps and are expressed as a

set of more or less formalized (design) models. Each of these models represents one concern of the targeted application independent of the rest of the issues in a simplified and readable form [Kappel et al. 2006]. This separation of design concerns facilitates a structured design and development process. Furthermore, some approaches also enable to (semi-)automatically generate an implementation for the targeted Web application based on its underlying design models.

The rest of this chapter provides a detailed overview of the most important existing approaches from both fields. When reviewing these approaches, a main focus is on the question of how they support different kinds of adaptation, such as personalization, ubiquity, or context dependency.

3.2 Component-based and Document-oriented Approaches

3.2.1 WebComposition Component Model

There already exists some work on component-based Web engineering (CBWE). Gellersen et al. discuss the problem of the coarse-grained implementation model of the Web and stress the importance of utilizing a more fine-grained approach [Gellersen et al. 1997]. Especially, they point out that there is a large gap between the fine granularity of existing design models and the coarse granularity of the Web's resource-based implementation artefacts (e.g. HTML documents). As a consequence, a fundamental problem arises when the design of a Web application is deployed to an implementation of file-based Web resources, not allowing for access the original design artefacts as entities anymore. To solve this problem, they suggest a component-based approach called the WebComposition Component Model [Gellersen et al. 1997].

WebComposition is based on an object-oriented model of Web software and aims at the hierarchical decomposition of Web applications into *components*. These are reusable elements defined on different abstraction levels. At a high level of abstraction a component might model a Web page or even a Web site. Further down the hierarchy components can represent fine-grained parts of pages, such as media elements, anchors, tables, etc. Whereas the leaves in the resulting component hierarchy are called *primitives*, the other components are called *composites*.

Components are defined by their states and behavior. The state of a component is described by a number of properties, each represented by an attribute-value pair. Properties can be referred to within any other property, thus allowing to express relationships (links) between components. The behavior of a component is defined by operations on its state. While component creators might define arbitrary operations, all components have to support the operations *setProperties*, *getProperties*, and *generateCode*. The latter one specifies how the state of a component can be mapped to its representation in the Web, for instance to HTML code.

For the XML-based definition of components the XML-based WebComposition Markup Language (WCML [Gaedke et al. 2000]) was introduced. Consequently, WCML documents act as virtual component stores consisting of a set of component descriptions. A WCML document is processed by the WCML compiler that maps the components described in that document to the file-based Web implementation model of a specific target language. Thus, the WCML compiler aims at analyzing the composition of components, resolving component references, creating a Web presentation according to the components' presentation behavior, and at passing those pages to a Web server. For more information on WCML the reader is

referred to [Gaedke et al. 2000].

Based on the notion of components (that are defined by their properties, states, and behavior), WebComposition utilizes a generic programming model, i.e. it does not prescribe what types or classes of Web components may exist, how they should generate (parts of) Web presentations, how their interfaces should look like, etc. Furthermore, it uses the prototype-instance-model [Ungar and Smith 1987] for modeling inheritance, i.e. every component might be used as a prototype for other components. This approach provides a simple and powerful means for reuse and code sharing, but also results in lacking type safety. Furthermore, this generic model also implies that developers have to take care of all important aspects of their concrete components, such as their specification, validation, as well as their automatic mapping to a concrete implementation provided by a specific Web document format.

As a generic model, the WebComposition approach allows to declare and create arbitrary kinds of reusable Web components. In [Graef and Gaedke 2000] Graef and Gaedke show how even Web applications with limited adaptation functionality can be created by reusable components using WCML. Still, WCML provides no inherent support for defining adaptable elements of content, navigation, presentation, and application behavior, nor does it provide mechanisms for describing different adaptation contexts, such as user preferences, device capabilities, etc.

Also inspired by the concepts introduced by WebComposition and WCML, this thesis will introduce a component-based declarative document model designated to develop adaptive Web applications. Yet, to represent the basic concepts and concerns characterizing adaptive hypermedia and Web-based systems, the proposed component model will provide a more explicit typing of components on a number of levels of abstraction, as well as a central focus on device, user, and context adaptation.

3.2.2 HMDoc

Westbomke et al. [Westbomke 2001, Westbomke and Dittrich 2002] propose HMDoc, an XML-grammar for the implementation and presentation of platform-independent structured hypermedia documents. It is based on the concepts of Tochtermann's hypermedia reference model [Tochtermann and Dittrich 1996] which were transformed into a declarative specification based on an XML document type definition (DTD). A hypermedia document described in HMDoc is called a *hyperdocument* and consists of so-called HMObjects as well as additional structuring elements (see Figure 3.1).

The elementary objects of an HMDoc document are so-called *components* aimed at the integration of *media objects* into a hypermedia presentation. A component contains exactly one media object and (optionally) a number of additional descriptive metadata attributes. On top of components so-called *document nodes* playing an information conveying role are specified. They can not only contain components but also aggregate other document nodes, so that an arbitrary deep hierarchy of document nodes is supported. Furthermore, in order to provide hypertext navigation functionality, so-called *links* are used. A *link* is characterized by its source and destination anchors, each of which may be (parts of) components, document nodes, or even entire hyperdocuments. A hyperdocument described in HMDoc is thus specified as a collection of document nodes, components, media objects, and links.

Besides the basic concepts of a hyperdocument (components, document nodes, and links), HMDoc also introduces so-called *structuring concepts*. Two structuring concepts are supported: 1) *subdocuments* (i.e. disjunct reusable subsets of a hypermedia document) and 2) *views*. The main goal of a *view* is to facilitate the user-specific presentation of a hyperdoc-

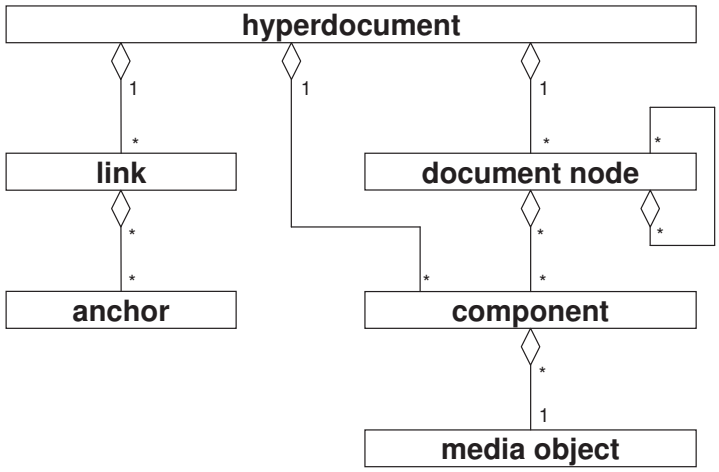

Figure 3.1: Basic structure of an HMDoc hyperdocument [Westbomke and Dittrich 2002]

ument by presenting only a restricted subset of it for a given user. Still, it is unfortunately not further specified under which conditions a view should be used in a given usage scenario (e.g. for a specific user or user group), nor is an explicit modeling of different usage contexts foreseen.

While HMDoc document descriptions are implementation and platform independent, Westbomke et al. [Westbomke and Dittrich 2002] also deal with aspects of their presentation and propose the usage of external XSLT-based techniques for this purpose. Still, the automatic adjustment of HMDoc documents to different user or device profiles (or models) or different output formats is not considered. Furthermore, though a number of requirements towards a visual authoring environment aimed at the intuitive creation of HMDoc documents are identified and mentioned, there is no appropriate implementation available, yet.

3.2.3 Intensional Hypertext

As a "complementary approach for adaptive and adaptable hypermedia" Wadge and Schraefel introduce the notion of Intensional Hypertext [Wadge and Schraefel 2001]. It is based on intensional logic, i.e. the logic of assertions and expressions, which "vary over a collection of contexts or possible worlds". The basic idea behind Intensional Hypertext is that authors produce HTML pages with extra markup (called *intensional tags*) which is aimed at explicitly delimiting parts that are sensitive in various ways to a given context. Thereby, a context is defined as sets of values for parameters which specify the current user profile as supplied by the current Web page URL and the latest user input. The resulting document format is thus a proprietary extension of HTML and is called Intensional Markup Language (IML [Wadge 2000])[3].

The language constructs of IML support basic adaptation mechanisms. *Conditional inclusion* allows to show or hide different versions of texts and HTML fragments based on context parameters. *Parameter substitution* means the inclusion of the values of context parameters into an HTML document. *Stretchtext* is text available in different levels of detail so that it can be adapted according to the current reader's expertise or interest. Similarly, *droptext* is a

[3]IML is a further development of Intensional HTML (IHTML [Wadge et al. 1998]), a former extension of HTML by intensional logic.

single block of text that can be made to appear or disappear separately, without affecting any other part of the document. Furthermore, IML also supports so-called *stereotype parameters* (i.e. parameters that have a discrete set of values, each of which represent a kind of common user profile) as well as *transversion links*. The latter are conventional HTML links extended with expressions that trigger context parameter updates that are automatically performed when a user follows these links. Based on this mechanism, parts of the context information can be changed during the user's browsing session.

The Web pages authored in IML are translated into a Perl-like language called ISE (Intensional Sequential Evaluator [Swoboda and Wadge 2000]). To generate the appropriately adapted individual pages at run-time, the Web server runs the ISE interpreter in the appropriate context. This interpreter produces HTML that, when displayed in the user's browser, is rendered into the desired adaptation of the requested page.

Though IML allows for quickly defining light-weight adaptation operations to be performed on Web documents, a main disadvantage of the approach is that it does not support for a clear separation of concerns such as content, layout, navigation or adaptation. Quite the opposite, since it is an extension of HTML, all these different aspects are hard-wired and intertwined in one single IML document, which makes its management and reuse quite complicated and also implies that there is no support for output formats other than HTML. Furthermore, so far there are no corresponding authoring tools available that would support the intuitive creation of IML documents.

3.2.4 CONTIGRA

The CONTIGRA [Dachselt et al. 2002, Dachselt 2004] research project[4] pursues the challenge to easily develop Web-based interactive 3D applications from reusable and standardized declarative components. Its main focus is on the identification and classification of 3D interaction elements (3D widgets) as well as on the creation of an XML-based architecture for their specification and composition.

In order to allow for component-based reuse, CONTIGRA introduces the concept of declarative 3D components. These are XML document instances describing reusable building blocks of 3D user interfaces that can be easily configured and put together to complex 3D scenes. They are specified by a number of XML markup languages (based on XML schema definitions), each declaring a specific aspect (geometry, composition, implementation, etc.) of 3D components (see Figure 3.2).

An instance document of the grammar CoApplication defines an overall 3D application in terms of typical scene parameters (such as lights and viewpoints) and a reference to a 3D root component containing the whole scene. This component is defined by two XML documents, one for its interface (according to the schema CoComponent), the other for its implementation (an instance of CoComponentImplementation). The interface document contains configurable high-level parameters defining a component's functionality as well as authoring and other meta information. The implementation document firstly contains a component graph which is a transformation hierarchy containing references to other 3D components. For all additional parts of the scene, which are not yet available as a reusable component, it secondly contains a scene graph. This is split into three parts for *audio*, *geometry*, and *behavior nodes*, usually as references to external scene graph files. At present X3D is used for the geometry, and the extended grammars Audio3D [Hoffmann and Dachselt 2003] and Be-

[4]The acronym CONTIGRA stands for "COmponent-orieNted Three-dimensional Interactive GRaphical Applications".

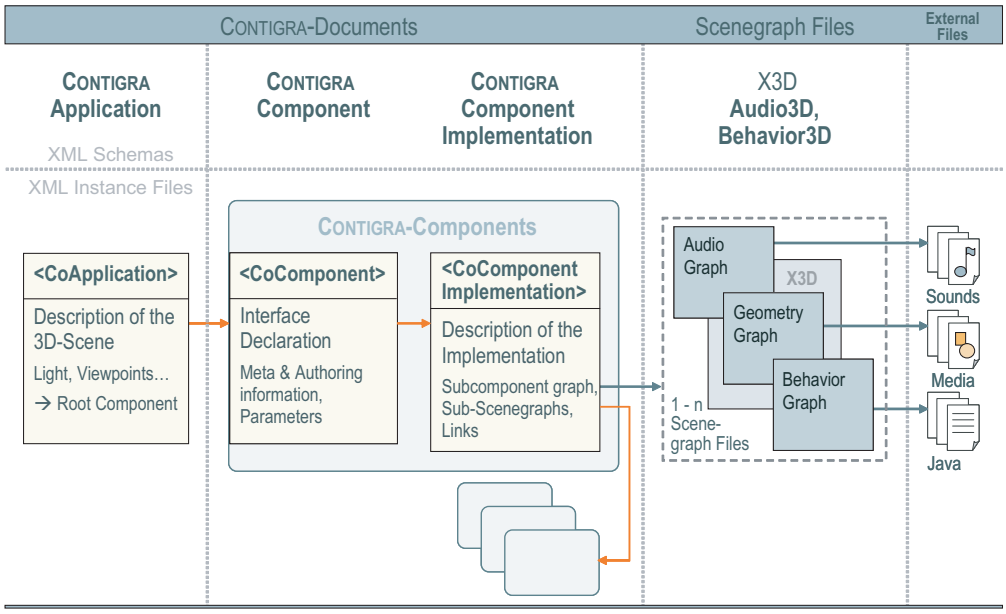

Figure 3.2: Overview of the CONTIGRA markup languages [Dachselt 2004]

havior3D [Dachselt and Rukzio 2003] for the definition of spatial audio and complex behavior nodes respectively. The third part of a CoComponentImplementation document consists of a separated link section connecting referenced parts.

In order to allow for the intuitive visual authoring of XML-based 3D applications from CONTIGRA components the CONTIGRABuilder [Dachselt 2004] was developed. It is based on an extensible repertoire of visual editor modules aimed at the visual composition of components as well as the configuration of various component properties (parameter, metadata, geometry, behavior, etc.). The applications created with the CONTIGRABuilder are described independently from proprietary 3D toolkits or APIs, therefore they can be automatically translated into target formats such as VRML97, X3D, OpenSG, MPEG-4 or Java3D by using either an internal object model (data binding) or a series of corresponding XSLT-Stylesheets.

CONTIGRA is a good example for the efficient application of reusable declarative implementation artefacts for Web-based application development. However, it focuses on the specifics of 3D user interfaces elements (geometry, behavior, etc.), not addressing the requirements of traditional two-dimensional hypermedia presentations. Furthermore, even though Dachselt mentions the importance of adjusting 3D applications to different users, contexts, and client devices [Dachselt 2004], CONTIGRA components do not provide inherent supoort for adaptation, yet.

3.2.5 CHAMELEON

The CHAMELEON project [Wehner and Lorz 2001] aims at developing formats and tools for reusable, adaptive courseware (courseware components) for Web-based eLearning systems. Courseware is represented in an XML-based document format called TeachML that distinguishes between four abstraction levels: *media objects*, *content units*, *didactical units*, and *structures* (see Figure 3.3).

On the lowest level, there are *media objects* that represent the atomic parts of TeachML

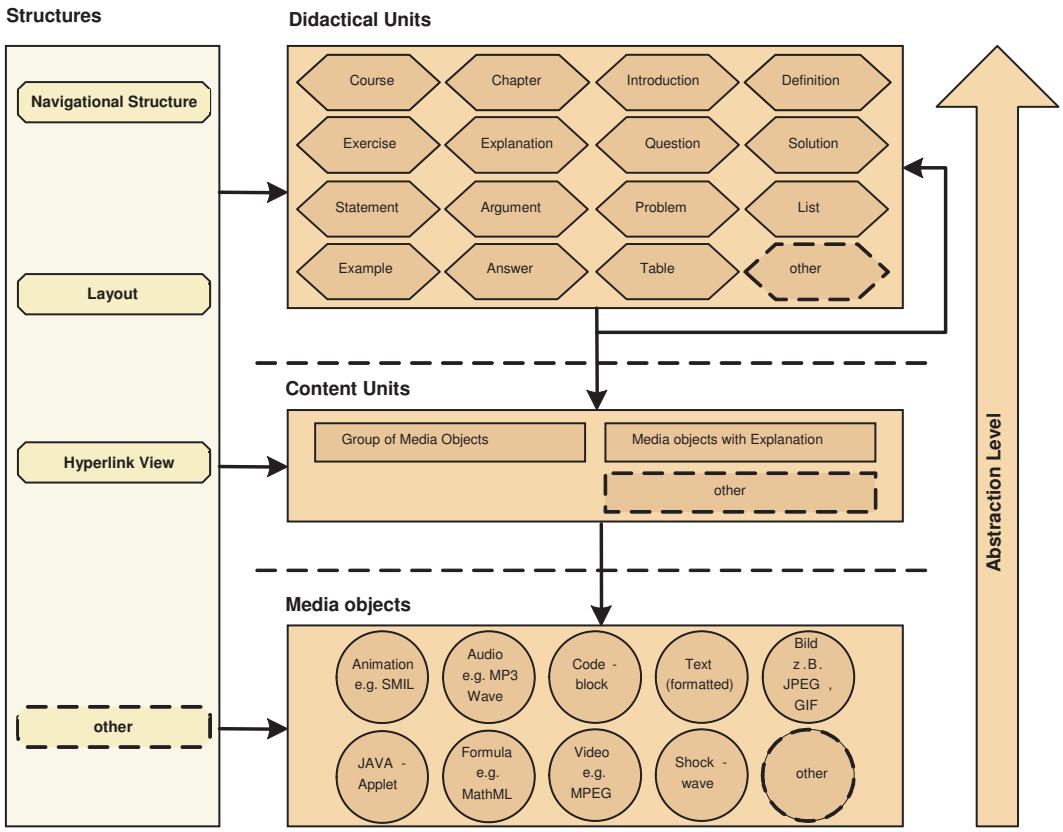

Figure 3.3: Overview of the TeachML document model [@CHAMELEON]

documents. They can either reference external media instances (e.g. images, audio or video files, etc.) or serve as direct containers for textual content such as plain text, source code, or formulas. On the second level, media objects are composed to so-called *content units*. They group media objects which belong together to transmit a common message to the learner, e.g. a *figure and its description* or a *set of media objects* (for instance consisting of several formulas, texts, and references). On the third abstraction level, *didactical units* are defined that play a well-defined didactical role in a TeachML-based eLearning course. They can not only contain content units but also other didactical units, i.e. an arbitrary deep hierarchy of didactical units is supported. A top-level didactical unit has the type *course*. Further predefined didactical unit types are *chapter*, *definition*, *line of arguments*, *example*, *exercise*, etc., but arbitrary author-defined extensions are also allowed.

While media objects, content units, and didactical units encapsulate content elements on different abstraction levels, the TeachML grammar also allows to define so-called *structures* on top of them (see the left side of Figure 3.3). *Navigational structures* are directed cyclic graphs that define navigational paths (in form of guided tours) through didactical units. *Layout structures* [Meißner et al. 2001] provide a facility for the spatial arrangement of content elements for a given output format (e.g. Web-based or print media). Finally, *hyperlink structures* allow to define hyperlink references between content elements on arbitrary levels.

For the visual authoring of TeachML documents the CHAMELEONBuilder was developed [Chevchenko 2003]. It is based on an extensible modular architecture and provides a number of editors plug-ins for graphically creating courseware components and interlinking

them to structures. Since the authoring tool for adaptive Web applications introduced in this thesis is based on the same plug-in architecture, it will be described in more detail in Section 5.2.

The levels and structures of TeachML allow to capture the main elements of educational hypermedia presentations. Moreover, the possibility to define typed components provides for efficient reuse and extensibility on all abstraction levels. However, an automatic adaptation of components to explicit student models or other context parameters (user preferences, device capabilities, environment context, etc.) has not been considered, yet. Note that the component-based document format to be introduced in Chapter 4 was significantly inspired by CHAMELEON, aiming at generalizing its component concept to a broader range of adaptive Web applications.

3.2.6 RIML

To reduce the development effort involved in building Web applications for mobile and other non-desktop devices, Ziegert et al. introduce the Renderer Independent Markup Language (RIML [Ziegert et al. 2004]). It is based on the "Author once – Display Everywhere" philosophy, i.e. the creation of Web content in a device independent markup language which then gets adapted to the special characteristics of the accessing device. Similarly to IML, RIML is not a stand-alone language, rather a custom extension to XHTML 2.0 which adds adaptation features such as pagination and device independent layout mechanisms. Furthermore, in order to support form-based interactions, the RIML profile also includes the XForms 1.0 language.

As a means for structuring content on a Web page RIML uses the `section` element of XHTML 2.0 [Axelsson et al. 2004]. As a consequence, the author of a RIML document is required to put all content that should be presented on the same screen into the same section. While sections might be nested, the innermost sections get never split up. The consequent use of sections allows to exploit the adaptation features provided by the RIML, i.e. its support for device independent layouts and pagination.

The layout module of RIML supports the specification of device (class) specific layouts. It defines a set of container types: *rows*, *columns*, *grids* as well as so-called *frames*. Whereas the containers define the overall structure of a layout definition, frames are used to fill the regions of the layout with content. This is accomplished by the assignment of frames to XHTML 2.0 sections, i.e. the content of a section is always rendered within the region of its associated frame.

A further adaptation mechanisms facilitated by RIML is pagination. In order to cope with the small display sizes of mobile devices, it aims at dividing the content assigned to a frame into multiple pages (by using sections as implicit splitting hints). To use this feature authors can annotate selected frames as "paginable". Furthermore, they can also control the pagination process by specific metadata attributes, e.g. by explicitly declaring the minimum width of a frame in pixels. When pagination occurs, so-called *navigation links* are generated between the split pages. Again, authors can use RIML-specific metadata to control the types and values of these links.

For the graphical creation of RIML documents the Consensus authoring environment was developed. It is implemented as a plugin of the Eclipse open source platform and comprises a set of so-called views and editors. For editing RIML documents a built-in XML editor is provided that supports for code completion and validation based on the RIML schemas. The *Frames Layout View* is a visual tool that allows a Web author to get a first impression

(preview) of how a document (based on an abstract RIML layout description) will look like on different platforms. Similarly, the *RIML Device Dependent View* provides an overview how a RIML document is paginated, i.e. how many pages are created, and what they contain. For more information on the RIML and its authoring tools the reader is referred to [Ziegert et al. 2004].

The RIML is well suited for the adjustment of textual (XHTML based) content to the limited displays of mobile devices, as was also demonstrated in the EU project Consensus [@Consensus]. Still, it does neither support basic adaptation techniques such as conditional inclusion of page fragments (or variants), nor non-textual media specific adaptations (such as the provision of media elements with quality alternatives or the replacement of images with video or textual content, etc.). Furthermore, the usage of XHTML does not allow for a clear separation of the concerns content, structure, navigation, presentation, and adaptation. Finally, the lacking support of current Web browsers for the standards XHTML 2.0 and XForms implies also some limitations to the ubiquitous applicability of the RIML.

3.2.7 The XiMPF document model

For the device independent publication of multimedia content Hendrickx et al. introduce in [Hendrickx et al. 2005] the XiMPF document model (eXtensible Interactive Multimedia Publication Format). It is fashioned after the MPEG-21 Digital Item Declaration (DID) model [ISO 2002], but uses a semantically richer set of elements to structure and annotate the presentation content.

The XiMPF document format defines a multimedia document as a tree-like hierarchy of composing *items* (see Figure 3.4). Each *item* combines a number of *presentations*, *descriptions*, *template instances*, and *subitems*. A *presentation* aims at the platform specific publication of an item's content (e.g. for the Web or for a set top box). It references a number of *description* elements that contain descriptive language constructs for specifying its structure (i.e. the subitems it uses), layout, synchronization, etc. *Description* elements further include references to subitems to precisely place them into a hierarchy. These references are resolved by so-called *template instances* that link a reference to a specific *presentation* in another *item*.

The XiMPF document model makes abstraction of item content. It allows to use atomic multimedia resources as alternatives for composite presentation fragments, for instance the combination of a picture and its corresponding textual caption might be an alternative for a video. Furthermore, XiMPF extensively uses existing W3C technologies (such as XHTML2, CSS, and SMIL) for structure, layout, and synchronization descriptions. However, it also introduces an own language (called XML Interaction Language) to specify the interactive behavior of a multimedia presentation.

The publication architecture of XiMPF is based on the XML publishing framework Cocoon [Ziegeler and Langham 2002]. It consists of an XML processing pipeline (aimed at resolving and adapting XiMPF documents to a given presentation version), a core engine, a resource and metadata database, as well as an adaptation service registry [Oorts et al. 2005]. The latter allows to register adaptation services for processing atomic multimedia resources like audio and video. This adaptation primarily concerns the adjustment of multimedia content to a client profile and is performed based on information gathered from the query string

[5]The dashed lines denote the inclusion of references to Description and TemplateInstance elements in Presentation elements. The arrowed lines represent the resolution of a UseItem element to a Presentation or Item through a TemplateInstance element [Hendrickx et al. 2005].

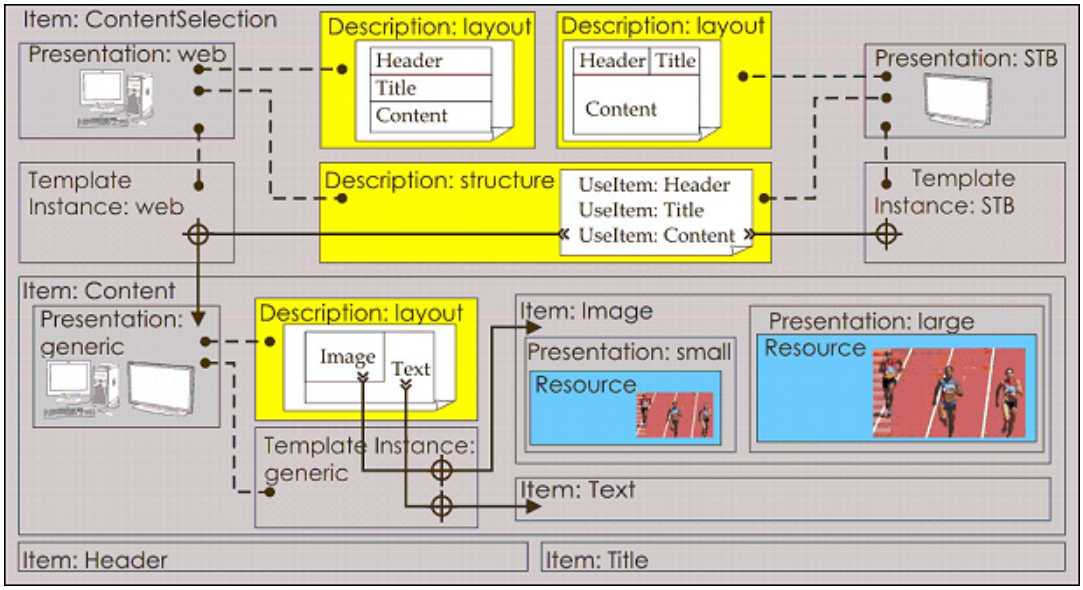

Figure 3.4: Schematic outline of an XiMPF document[5] [Hendrickx et al. 2005]

of HTTP requests.

The main strengths of the XiMPF document model are the reuse of both media resources and presentational information at a fine level of granularity, as well as the automatic adaptation of both singular and composed media content. However, while its main focus lies on the presentation of multimedia content on different presentation platforms, it does not explicitly address basic aspects of hypermedia adaptation (e.g. the adjustment of a Web site's logical or navigational structure), nor other context parameters (e.g. dynamic user information or environmental data). Furthermore, there are no graphical tools aimed at the intuitive visual creation and manipulation of XiMPF documents (and descriptors) available, yet.

3.2.8 Portlets as Portal Components

The recently emerged notion of Web portals indicates Web sites that provide a comprehensive entry point for a large array of resources and services. A portal application typically contains different modules (e.g., news, free e-mail services, search engines, online shopping, chat rooms, discussion boards, or links to other sites) and also provides some kind of per-user customization for these services. Hepper defines a portal as a "Web-based application that provides personalization, single sign-on, and content aggregation from different sources, and hosts the presentation layer of information systems [Hepper 2004]". According to his definition, a portal is composed of pluggable interface components that represent a certain application functionality, generate dynamic Web content, and thus enable modular Web applications.

In the recent years, different incompatible APIs for portal components have been introduced by various vendors. To overcome the problems arising from their missing interoperability, the Java Portlet Specification JSR 168[6] [Abdelnur et al. 1999] was proposed. According to JSR 168, a portal consists of *portal pages*, each being an aggregation of so-called *portlets*. A portlet is a Java-based Web component that processes requests and generates dynamic

[6]The acronym JSR stands for Java Specification Request.

markup in form of HTML, XHTML, or WML fragments [Hepper 2004]. The fragments generated by the aggregated portlets form a complete Web document. Portlets can store persistent data for a specific user and also maintain temporary session information. Their life-cycle is managed by a so-called *portal container* that provides them with the required runtime environment. Besides local portlets, a portlet container can also run remote portlets by using the Web Services for Remote Portlets (WSRP [Allamaraju and Brooks 2005]) protocol.

Each JSR 168 compatible portlet must implement the so-called *portlet interface* defining the basic portlet life-cycle. This life-cycle comprises the phases of *portlet initialization*, *request handling*, and *portlet destruction*. The request handling phase is further divided into the two categories *action handling* and *rendering*. The latter allows the portlet to produce markup depending on the portlet's state information, backend data, its so-called *portlet mode*, and *window state*. The portlet mode indicates the function a portlet performs and can be either a custom mode or one of the three standard modes `VIEW` (the portlet generates markup), `EDIT` (the portlet lets a user customize it), or `HELP` (the portlet provides help information). The availability of portal modes may be restricted (i.e. personalized) to specific user roles of the portal. Similarly, a portlet's window state is an indicator of the amount of portal page space that will be assigned to the content generated by a portlet, i.e. a portlet might produce different markup for each window state. The generation of markup is performed by the portlets' *doView* method, mostly based on the invocation of a JSP template producing HTML fragments. Unfortunately, this rather low-level programming model does not allow to specify application and adaptation concerns in a more higher-level declarative way, nor does it provide type safety for portlet composition.

Besides the portlet mode and the window state, portlets can adapt themselves to the actual portal that calls them based on the so-called `PortalContext` containing information on the portal vendor, the portal version, and specific portal properties. Furthermore, for the sake of personalization, they can also access user profile information according to the W3C recommendation P3P (Platform for Privacy Preferences [Cranor et al. 2002]). However, the data provided by P3P contains mainly (static) user identification and contact information (e.g., name, post address, phone number, email). Consequently, the provided personalization services are mostly restricted to the user-specific structuring (inclusion or exclusion) of portlets as well as the individualization of their presentation style (in terms of colors and style elements). Other kinds of adaptation, such as media or hypermedia navigation adaptation, are not inherently supported and must be programmed manually by portlet programmers.

3.2.9 Active Documents

Whereas the aforementioned approaches represent concrete document formats or component models, Aßmann proposes general requirements and architectural styles for declarative component-centric document models. He introduces in [Aßmann 2005] the concept of *active documents*. These are documents that contain "both data and software, data and macros, or data and scripts", and that can be manipulated interactively. Typical examples for active documents are spreadsheets, office documents containing macros, but also Web documents comprising dynamic elements such as scripts, applets, etc. The main characteristic of an active document is that it contains *derived components* (e.g. HTML fragments) that are automatically generated (derived) from a set of *base components* (e.g. HTML templates) by a so-called *embedded software* (e.g. a template engine). That is to say, "an active document exploits the power of programming to represent document content more concisely".

To effectively cope with the complex engineering process of active documents, Aßmann proposes to explicitly discern their architecture, i.e. to provide an architectural language

for them. From frequent problems in engineering active documents he derives three main requirements: 1) support for *invasive composition operations*, 2) *transconsistency*, and 3) *staged architectures*.

Invasive composition operations: Invasive composition operations are operations that allow to embed document fragments into document templates [Aßmann 2003]. Two kinds of operations can be distinguished: *parameterizations* and *extensions*. The former means that code templates carry *slots* that can be filled with other code fragments, so code templates are instantiated towards executable components. The latter denotes that code templates also carry so-called *hooks* (extension points), which can be extended with other fragments.

Transconsistency: The concept of transconsistency provides "hot updates" in an active document. This means that every change to (parts of) an active document is automatically propagated to all dependent parts immediately. Transconsistency is an extended form of *transclusion*, a basic operation in hypertext systems. Transclusion ensures that whenever a node, which is included into several nodes, changes, all embedding components also change immediately. Transconsistency extends transclusion by supporting arbitrary operations (besides embedding). A typical scenario is the dynamic generation of fragments of an HTML document (derived components) based on XML documents (base component) by means of XSLT stylesheets (embedded software). Thus, whenever a base component is edited, all derived components are also changed. Furthermore, approaches such as applet-servlet interaction in Web form processing can be also described as generalizations of transconsistency.

Staged Architectures: In order to address the specifics of Web-based systems the notion of *staged active documents* is proposed. These are active documents that are processed in a series of so-called *stages*, each producing code for the next stage. A *staged architecture* is a sequence of n stages, where the nth stage produces the final document. Such an architecture is prevalent for dynamic Web-based applications, where (sequences of) server-side processing operations are optionally followed by the execution of client-side scripts or applets. Thereby, each processing step (stage) might be both invasive and/or transconsistent.

After identifying architectural styles for active documents, Aßmann defines a hypothesis for their composition. According to this, a compositional technique for active documents relies on four concepts: 1) explicit architectures for both software and documents (including component models), 2) invasiveness, 3) staging, 4) and transconsistency. With these concepts, he explains the architecture of many document processing applications, especially of Web-based systems [Aßmann 2005].

Aßmann also denotes that a multitude of today's Web-based architectures rely on component models for the software and data components of active documents. Furthermore, a number of them also supports invasive operations (e.g. by expanding HTML templates), transconsistency, and staged architectures[7]. Nevertheless, as a main shortcoming of existing techniques he identifies the lacking support for typed composition operations and explicit architecture descriptions, which leads to "maintenance headache and a major cause for future legacy systems". To overcome this problem, he proposes fragment-based component models that are controlled by meta models or schemas, respectively.

[7]Note that the publication of WCML, IML, RIML, and XiMPF documents is based on staged architectures.

We remark that such an explicit fragment-based component model for adaptive Web presentations has been developed within the scope of this dissertation project and will be presented in Chapter 4. Though published prior to Aßmann's work on active documents, it provides support for the three main requirements he mentions (i.e., invasiveness, transconsistency, and staging).

3.2.10 Summary and Comparison

This section dealt with existing component-based and document-oriented Web engineering approaches regarding their applicability for personalized, adaptive Web applications. It was shown that they not only address different application areas, but also differ in their (extent of) support for important engineering aspects[8], such as reusability, extensibility, platform-independence, adaptability, etc. Based on the criteria listed below, this subsection provides a comparison of the reviewed solutions.

1. **Application area:** Is a particular component model designated to a specific application area (e.g. Web-based 3D applications or eLearning systems) or is it a general approach?
2. **Explicit separation of concerns:** Is an explicit separation of different Web application concerns (e.g. content, structure, navigation, presentation, etc.) provided?
3. **Reusability:** To what extent (e.g. at which granularity) is the reuse of components (or document fragments) provided?
4. **Composability:** Is it possible to aggregate reusable components (or fragments) to composite components that can be again subject to further composition?
5. **Device/platform independence:** Is the approach/technique based on a particular platform, Web output format (e.g. HTML), or specific end device; or does it allow a platform- and device-independent specification of components?
6. **Automatic presentation generation:** Is an automatic transformation of components to a given Web output format provided or does this require additional efforts from developers[9]?
7. **Built-in adaptation support:** Does the component (or document) model provide inherent support (e.g. in form of buil-in language constructs) to specify the adaptation behavior of components? Which kinds (or aspects) of hypermedia adaptation (see Chapter 2.2) does it cover?
8. **Template support:** Is there support for data-driven Web or hypermedia applications? Is it possible to define component templates (or document/fragment templates) that can be dynamically extended (filled out) based on a query to a dynamic data source? Note that this aspect corresponds to Aßmans requirement towards invasive composition (see Section 3.2.9).

[8] The aspects listed here also serve as basic requirements towards the solution proposed in this work (see Chapter 4).

[9] Note that while HMDoc or WCML allow to describe Web or hypermedia applications in a platform-independent way, authors of such documents/components are required to take care of the platform-specific presentation of their components themselves. This means additional efforts in comparison to e.g. CONTIGRA or CHAMELEON that support an automatic code generation for some given output formats.

9. **Interoperability with standards:** Does a particular approach make use of (or is it based on) on existing internet, W3C, or industry standards?

10. **Tool support:** Is the approach optimally supported by corresponding visual tool support?

Based on these criteria, Table 3.1 provides a tabular comparison of the presented approaches. Note that while some rows contain a short textual explanation or comment, some only provide a "rating" based on the following rating scheme: *no support* (-), *limited support* (+), *good support* (++).

As can be seen, the majority of existing approaches (except for CONTIGRA and CHAMELEON) addresses Web (or hypermedia) applications in general and is thus not designated to a specific application area. Furthermore, most solutions are based on XML technology and also provide some support for platform independence by abstracting from a concrete Web output format. However, only a few approaches allow for the automatic generation of a Web presentation in a variety of output formats. Moreover, besides the separation of content and presentation, there is only limited support for explicitly distinguishing further concerns, such as navigation, semantics, behavior, or adaptation. Similarly, only a few solutions (WCML, CONTIGRA, and CONTIGRA) provide the advantages of traditional component models, among them fine-grained reuse, composability, and extensibility.

As the biggest shortcoming of all investigated approaches, one can consider their lacking support for adaptation. Only a few of them supported (very limited) aspects of content and presentation, the majority of the basic hypermedia adaptation techniques classified in Section 2.2 is not addressed at all. Similarly, none of the described component models contains an explicit context or user model, and only a few of them are supported by visual development tools. Finally, hardly any approach supports component (or document fragment) templates explicitly, i.e. only limited authoring support for dynamic Web Information Systems is provided.

3.3 Model-based Web Design Methods

Recently, a number of model-based Web design methods for hypermedia and Web applications have been developed. Among the most significant contributions we mention the Relationship Management Methodology (RMM [Isakowitz et al. 1995]), the Object Oriented Hypermedia Design Model (OOHDM [Schwabe et al. 1996]), the Web Site Design Method (WSDM [De Troyer 2001]), the Web Modeling Language (WebML [Ceri et al. 2000]), and the Hera specification framework [Vdovjak et al. 2003]. Even though utilizing different formalisms and notations, a common characteristics of all approaches is to distinguish between the *conceptual model* describing the application domain, the *navigational model* specifying the (abstract navigational) structure of the hypermedia presentation, and the *presentation model* specifying the rendering of navigation objects (layout). Some methodologies extend these basic models by additional ones concerning further aspects of a Web application, such as user interaction or different kinds of adaptation (personalization, device independence, localization, etc.). Furthermore, selected approaches also provide visual authoring tools support for creating their models as well as facilities for the (semi-)automatic generation of a running implementation based on these models.

There are different criteria to classify hypermedia design methodologies. One possibility is to distinguish between the modeling techniques and formalisms they utilize. According to this aspect, Kappel et al. [Kappel et al. 2004] distinguish between *data-oriented* (e.g.

	WCML	**HMDoc**	**IML**	**CONTIGRA**	**TeachML**	**RIML**	**XiMPF**	**Portlets**
application area	Web apps.	hypermedia	Web apps.	3D Web apps.	Web-based eLearning	Web apps.	Web apps.	Web apps.
explicit separation of concerns	-	content and navigation	-	geometry, behavior, audio	content, semantics, presentation, navigation	content, presentation	content, presentation	-
reuse support	++	+	-	++	++	-	++	+
composability	++	++	-	++	++	-	++	+
extensibility	++	+	-	++	++	-	++	-
device/platform independence	+	+	-	++	++	+	++	-
automatic presentation generation	-	-	HTML	X3D, MPEG-4	XHTML, PDF	XHTML	XHTML	-
built-in adaptation support	-	-	+	-	-	+	+	+
adapt. navigation	-	-	-	-	-	-	-	-
adapt. content	-	-	+	-	-	-	+	-
adapt. presentation	-	-	-	-	-	+	+	+
template support	-	-	-	-	-	-	-	+
tool support	-	-	-	++	+	+	-	+
interoperability with standards	XML	XML	-	X3D, MPEG-4	XML, SCORM	XHTML XForms	XML, MPEG-21 DID	JSR 168, WSRP

Table 3.1: Comparison of component-based and document-centric Web engineering solutions

RMM, Hera, WebML, SiteLang [Thalheim and Düsterhöft 2001]), *hypertext-oriented* (such as HDM [Garzotto et al. 1993], WSDM), *object-oriented* (e.g. OOHDM, UWE [Koch et al. 2001], OO-H [Gómez et al. 2001], OOWS [Pastor et al. 2003]), as well as *software-oriented* (e.g. WAE [Conallen 2000]) methodologies. Frasincar [Frasincar 2005] further identifies design methods for Semantic Web Information Systems (SWIS) that make use of Semantic Web technology (XWMF [Klapsing and Neumann 2000], OntoWebber [Jin et al. 2001], Hera, OntoWeaver [Lei et al. 2005], SHDM [Schwabe and de Moura 2003], SEAL [Maedche et al. 2003], etc.). Another possible distinction can be made depending on the order in which different aspects of the resulting application are specified. Most existing approaches are *content-driven*, i.e. they begin with the modeling of a Web application's underlying content, which is followed be the proper specification of its hypertext structure and user interface. However, there are also *presentation-driven* methodologies that start with the design of the user interface of a Web application. Similarly, *task-driven* (or *audience-driven*) methodologies are based on a detailed specification of user tasks to be supported by the application.

The rest of this section provides an overview of the most relevant existing approaches for designing hypermedia and Web-based systems, namely RMM, OOHDM, WSDM, WebML, and Hera. Thereby, a special focus is on the question of how they support the specification of different kinds of adaptation. Note that besides the Web design methods discussed here there exist also other approaches (see above). Still, they either do not address adaptation at all or do not support techniques that are not provided by the methods mentioned here. For a more detailed overview of Web design methods the reader is referred to [Murugesan and Deshpande 2001, Frasincar 2005, Casteleyn 2005].

3.3.1 Relationship Management Methodology (RMM)

One of the first approaches aiming at the structured design of data-centric hypermedia applications is the Relationship Management Methodology (RMM [Isakowitz et al. 1995]). It is based on the consideration of a hypermedia application as a system that manages information objects and their relationships. RMM distinguishes between four design steps: *E-R design*, *application design*, *user interface design*, and *construction/testing*.

The focus of *E-R design* is to specify the data managed by the hypermedia application in terms of entities and their associative relationships [Chen 1975]. Entities have attributes describing the characteristics of the data they represent. Similar to database modeling techniques, there are one-to-one and one-to-many relationships.

Application design aims at grouping attributes to so-called *slices*. A slice is a meaningful presentation unit representing a group of attributes that have to be shown together. Slices can be both aggregated as well as interlinked by using navigation primitives. RMM allows for different navigation primitives (called access structures) such as *indices*, *guided tours*, *links*, *groupings*, etc.

The next step is called *user interface design* and aims at describing the design of screen layouts for every element (slice) defined at application design. This includes button layouts, the appearance of nodes and indices, and the location of navigational aids. However, instead of providing a more formal notation, RMM suggests to use the "paper and pencil" strategy for this phase. Finally, the last step (called *construction and testing*) focuses on the implementation and testing of the resulting application based on traditional software engineering methods.

Díaz and Isakowitz propose in [Díaz et al. 1995] the design of RMCase, a tool to support RMM. The proposed tool offers so-called *contexts* (or views) corresponding to the above

described design phases. Whereas there are visual contexts designed for drawing the E-R model and the application model, the user interface has to be specified by creating HTML templates, mostly based on some third-party HTML editor. Furthermore, the created HTML templates are assumed to have appropriate "slots" which can then be populated with data at run-time.

While RMM is one of the first approaches aimed at a clear separation of concerns in hypermedia design, it does not address adaptation, personalization, or device independence. Furthermore, no output formats different than HTML are supported.

3.3.2 Object-Oriented Hypermedia Design Method (OOHDM)

The Object-Oriented Hypermedia Design Method (OOHDM) [Schwabe et al. 1996] is a methodology aimed at the design of complex hypermedia applications. It is based on well-known concepts of object-oriented application development (OMT) as well as on the Hypertext Design Model (HDM [Garzotto et al. 1993]). Recently, the concepts of OOHDM were adopted to the context of the Semantic Web [Berners-Lee et al. 2001] in the form of the so-called Semantic Web Hypermedia Design Method (SHDM [Schwabe and de Moura 2003]). Though it differs from OOHDM by using Semantic Web technology (RDF(S) and OWL) for expressing its models, the two methods are conceptually the same. They define five development phases: *requirements gathering*, *conceptual design*, *navigational design*, *abstract interface design*, and *implementation* (see Figure 3.5).

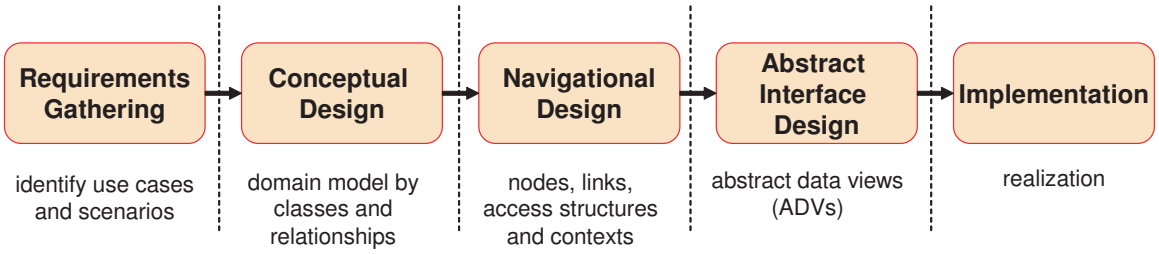

Figure 3.5: OOHDM/SHDM overview

The *Requirements Gathering* phase identifies the users of the system and the activities they would like to perform with the system based on scenarios and use cases. For each use case, OOHDM/SHDM introduces a user interaction diagram, which graphically represents the interaction between the user and the application. Subsequently, in order to validate each use case, the designer might interact with users to obtain feedback and optionally adjust interaction diagrams.

The *Conceptual Design* phase specifies the overall application domain based on classes and their relationships. The used notation is based on UML class diagrams, but it provides additional features such as multiple valued attributes as well as explicitly directed relationships. In SHDM this notation was replaced by RDF(S) and OWL [Patel-Schneider et al. 2004].

The *Navigational Design* phase defines a hypertext view on top of the conceptual model. It is expressed by a *Navigational Class Schema* and a *Navigational Context Schema*. The former specifies the objects of a hypermedia application through which users can navigate. These can be *Nodes* that group class attributes from the conceptual classes, *Links* between these *Nodes* as well as *Access Structures* that provide different navigation possibilities such as indices, menus or guided tours. The *Navigational Context Schema* allows to restrict the navigation spaces accessible to specific user groups by grouping navigational objects to so-

called *Contexts*.

The following phase is called *Abstract Interface Design*. It specifies the application's user interface by using *Abstract Data Views (ADV)* that define the interface appearance of navigational classes, access structures, menus, buttons, etc. Abstract Data Views are formal, object-oriented models of interface objects, allowing to defined the appearance of navigational objects in a high-level manner [Cowan and de Lu 1995].

Finally, the *Implementation* phase aims at the realization of the designed application. In this phase, the designer has to map the navigational and abstract interface models into concrete objects available in the chosen implementation environment. The model generated after performing previously defined activities can be implemented in a straightforward way using many of the currently available hypermedia platforms such as Hypercard, Toolbook, Director, HTML, etc. [Schwabe et al. 1996].

In order to explicitly address different kinds of users, Rossi et al. extended OOHDM by different personalization mechanisms [Rossi et al. 2001]. This personalization is expressed by introducing the concept of the *user class* as part of the application's conceptual model. Attributes of the user class can be subsequently used to refine (or parametrize) the results of navigational design, in order to adjust the information that is shown to the user (e.g. by offering a personalized price reduction) or to select or recommend links that are more relevant to him. However, OOHDM makes no further assumptions on the design and implementation of the corresponding link recommendation algorithms. Furthermore, only personalization examples concerning navigational design are discussed. Important aspects of personalization such as device-independence, presentation layer adaptation, as well as dynamic adaptation (according to a continually changing user or context model) are not addressed.

For developing Web applications using OOHDM different tools have been developed. As one of the first tools the OOHDM-Web environment was introduced [Schwabe et al. 1999]. It provides three interfaces: the *authoring environment* for creating navigation schemas based on the generation of corresponding database definitions, the *browsing environment* for specifying HTML templates corresponding to ADVs, and the *maintenance environment* for specifying interfaces for inserting instance data. Furthermore, it is also supported by a CASE environment allowing to describe the conceptual, navigational, and interface models using the OOHDM notation. Another implementation is OOHDM-Java2 [Jacyntho et al. 2002] which is based on J2EE (Java 2 Enterprise Edition) technology and supports OOHDM models that are extended by a business model as a generalization of the conceptual model and the application's transactional behavior. In this implementation OOHDM models are stored as XML documents and the page templates are defined in JSP (Java Server Pages).

Finally, besides the aforementioned "native" implementations, we also mention a different solution aimed at the mapping of OOHDM design artefacts to a component-based implementation. Segor and Gaedke propose in [Segor and Gaedke 2000] a number of heuristic implementation rules to map high-level OOHDM design specification to WCML components (see Section 3.2.1). The usage of such a fine-granular implementation base provides for better traceability and maintainability of the final implementation code. A similar approach allowing an even automated mapping of high-level design artefacts to a component-based implementation will be described in Section 5.3 of this thesis.

3.3.3 Web Site Design Method (WSDM)

The Web Site Design Method (WSDM [De Troyer 2001, Casteleyn 2005]) is an audience-driven Web design methodology. This means that it starts with an explicit modeling of a

Web application's users, their tasks, and their requirements, and uses this information to specify the conceptual, navigational, and presentational aspects of the resulting Web site. An overview of the design steps of WSDM is depicted in Figure 3.3.3.

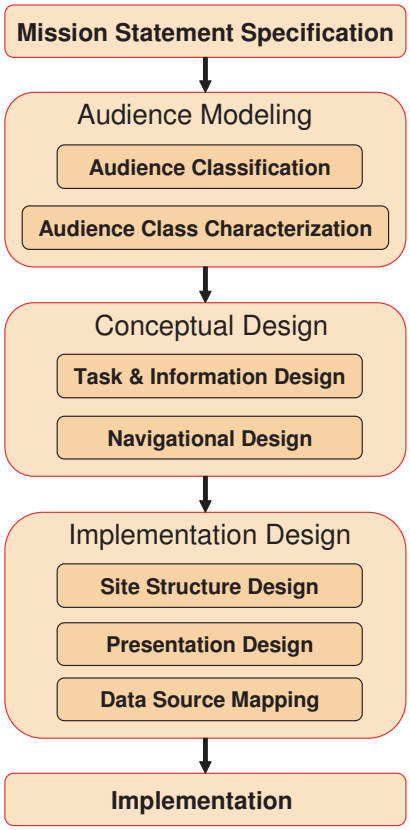

Figure 3.6: WSDM overview

In the first phase of WSDM the so-called *mission statement* is expressed. It specifies the purpose, the subject, and the targeted users of a Web site, and is formulated in natural language. It is followed by the two-step *audience modeling* phase that aims at identifying the different types of visitors of the Web site, as well as their requirements and characteristics. In the first step, *audience classification*, the different kinds of users (*audience classes*) are identified and ordered into a so-called *audience class hierarchy*. Thereby, an audience class comprises visitors with similar functional and informational requirements. In the second step, *audience class characterization*, the characteristics of the different (previously identified) audience classes are specified in more detail. These characteristics are later taken into account when deciding how to present information to these particular visitors.

The next phase of WSDM concentrates on the *conceptual design* of the site. Again, it is divided into two substeps: *task and information design*, and *navigational design*. In the *task and information design* substep the tasks to be performed by each audience class are modeled. For this purpose a modified version of the Concurrent Task Tree (CTT [Paterno et al. 1997]) notation is utilized, which allows to hierarchically order and condition tasks as well as to specify temporal relations between them. Furthermore, for each identified task a conceptual data model (*object chunk*) has to be constructed, which exactly describes the information/-functionality that is needed to fulfill this task. Object chunks are modeled by using a slightly

modified version of the Object Role Modeling (ORM [Halpin 2001]) technique. The role of the *navigation design* step is to define the navigational structure of the Web site and to model how the different audience classes can navigate through it. The central navigation entities are nodes that represent units of information or functionality. The information or functionality represented by a node is denoted by connecting the node to one or more chunks. Furthermore, nodes can be connected by links. Four different types of links are supported: *structural links*, *semantic links*, *navigation aid links*, and *process logic links* [Casteleyn and De Troyer 2002]. Structural links provide the actual structure of the information and functionality being offered on the site. Semantic links represent semantic relationships that exist (in the universe of discourse) between the concepts represented by the nodes involved. Navigation aid links are put on top of the existing structural link structure, and are aimed to better facilitate navigation for the visitor. Finally, process logic links connect two or more nodes to express part of a workflow or an invocation of an (external) functionality.

The *conceptual design* phase is followed by the *implementation design*, which again consists of three sub phases: *site structure design*, *presentation design*, and *data source mapping*. During *site structure design* the nodes defined at *navigation design* are grouped into so-called pages. *Presentation design* describes the layout of those pages. Subsequently, the *data source mapping* sub phase aims at creating mappings between the object chunks and the actual data to be presented.

Finally, taking as an input the object chunks, the navigation design, and the implementation design, the actual implementation can be generated. This transformation can be performed automatically and was realized in a prototype. For a more detailed description of the WSDM design phases and their corresponding models the reader is referred to [De Troyer 2001, Casteleyn 2005].

To specify the (run-time) adaptive behavior of a Web site at design time, Casteleyn introduces the Adaptation Specification Language (ASL [Casteleyn et al. 2003, Casteleyn 2005]). The main idea behind this approach is the automatic reorganization of a Web site based on user access data that can be collected at run-time. Still, instead of being personalized for individual users based on their (individual) browsing patterns, the Web site adapts itself to common user browsing patterns by gathering access information from *all users*. With the Adaptation Specification Language, the designer has a means to specify when certain adaptation should be applied (i.e. the *adaptation policy*) and which adaptation should be performed (i.e. the *adaptation strategy*).

ASL is an ECA[10]-based rule language: it uses events and conditions that trigger actions. Events can be user events (such as starting or ending a session, clicking on a link, loading a Web site element), system events (e.g. the initialization of the Web site itself) or time events (specifying the elapse of a certain time interval). Conditions can be defined on the basis of constants and variables. Actions are transformations of the Web site's structure (i.e. manipulation of object chunks, nodes, pages) and navigation (e.g. removing, adding, moving links). Adaptation of content and presentation (e.g. based on client device capabilities or other context information) have not been considered, yet.

3.3.4 WebML

WebML (Web Modeling Language [Ceri et al. 2000, Ceri et al. 2003b]) was developed at the Politecnico di Milano and is a "visual language" aiming at the specification of data-driven Web applications. Its models utilize a graphical representation but can be also serialized to

[10]ECA stands for event-condition-action rules [Dayal 1988].

an XML-based notation.

The basic modeling phases of WebML strongly resemble those introduced by OOHDM. The *data design* phase of WebML specifies the data model of the Web application by means of E-R-diagrams [Chen 1975]. It is followed by the *hypertext design* phase that is concerned with the construction of a coherent navigation model for the Web site based on the concept of content units and links. Content units are the basic elements that can be shown on a Web page. They can either publish information from a data source or represent forms with which content can be entered. Content units can be aggregated to more complex navigational elements such as *pages*, *page areas* (group of pages logically belonging together) or even whole *site views*. Finally, the *implementation* phase aims at mapping the data schema to a data source as well as at implementing the WebML pages by mapping them to JSP templates. However, WebML does not not include a specific model for expressing presentation at the conceptual level and hides the presentation in application specific XSLT stylesheets. The drawback of this approach is that system maintenance becomes difficult, since these stylesheets have to be implemented for each specific output device and format[11].

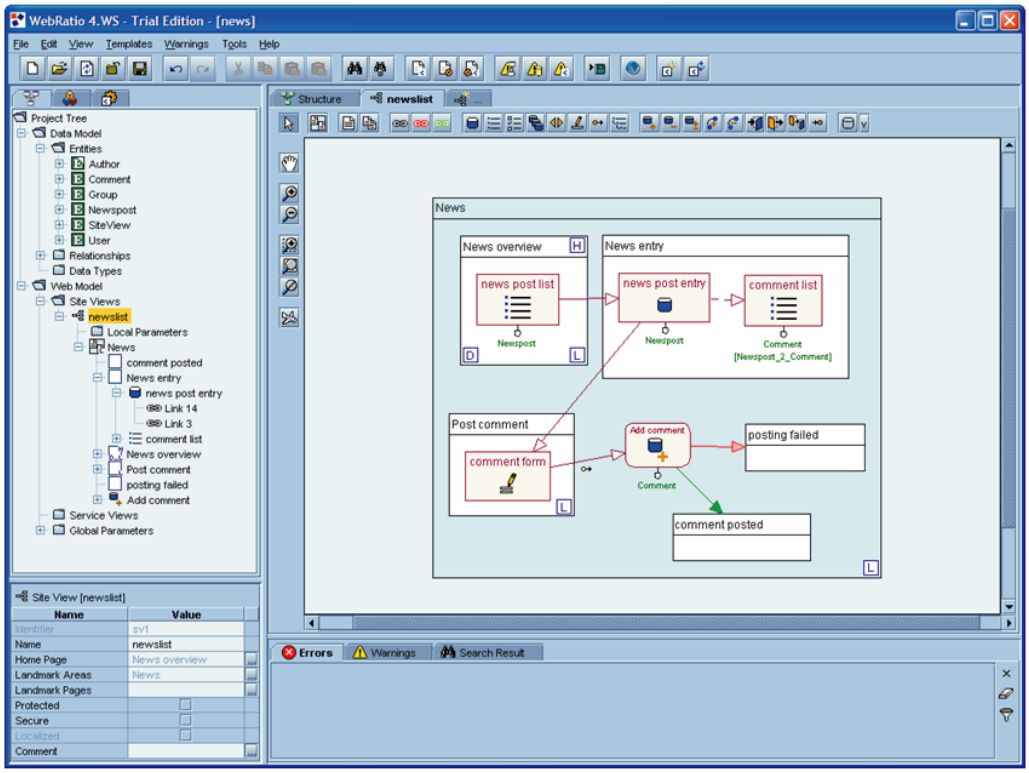

Figure 3.7: WebRatio site view example [@WebRatio]

In order to support personalization, WebML includes an explicit notion of *groups* and *users* as parts of a Web application's data model. The standard profile of a user includes identification information, login, trace information (visited pages and time of visit), and group membership, but is extensible to fit a given application domain. Groups contain individual users that are somehow related (e.g. children) and can thus be associated with dedicated site views. Designers can utilize ECA-based business rules for computing and manipulating such

[11] As an alternative solution, the approach proposed in this dissertation will utilize abstract layout descriptions that can be automatically transformed to a given Web output format(see later in Section 4.3.2).

user specific information [Ceri et al. 1999]. These allow to classify users in user groups, to manage user-specific information (e.g. a shopping cart), or to push information to users (e.g. on new purchase opportunities).

Furthermore, in [Ceri et al. 2003a] the notion of a *context model* is also introduced, allowing to describe *context entities* (e.g. device or location) associated to groups or users. Using these additional models designers can specify both context-aware pages (i.e. pages adapted according to context attributes) as well as different site views for specific users, groups, and contexts. Nevertheless, the corresponding adaptation conditions primarily intervene on the level of the hypertext model, adaptations in the presentation layer have not been considered, yet.

As a visual environment supporting the WebML methodology WebRatio [@WebRatio] was introduced (see Figure 3.7). It is without doubt the most mature available CASE tool for model-based Web application development that is also used commercially. However, the tools provided by WebRatio still do not cover the personalization and adaptation aspects provided by the WebML models.

3.3.5 Hera

Hera [Frasincar et al. 2002, Vdovjak et al. 2003] is a model-based methodology for the design and structured development of Adaptive Web Information Systems (AWIS). It has its origins in the RMM design methodology (see Section 3.3.1), i.e. it focuses on the design of Web Information Systems from a data-oriented perspective. Hera extends the concepts of RMM by a number of additional modeling features, such as personalization, adaptation, or user interaction. Furthermore, it uses Semantic Web technologies (RDF and RDFS) for expressing the different models that describe an AWIS. As formerly mentioned, Frasincar refers to Hera as a SWIS (Semantic Web Information System) methodology [Frasincar 2005].

Similar to the other methods described above, Hera distinguishes between three aspects of WIS design: the semantic aspect, the navigational aspect, and the user interface aspect. Each of these aspects is specified in form of a model: the Conceptual Model (CM) describing the data of the application domain, the Application Model (AM) specifying the application's navigational structure, and the Presentation Model (PM) specifying its user interface.

In addition to these basic models, Hera puts a main focus on the specification of adaptation in a WIS. As described in [Frasincar et al. 2002], it considers adaptability (or static adaptation) and adaptivity (dynamic adaptation)[12]. Nevertheless, in comparison to most other methods mainly focusing on navigation adaptation, the adaptation design in Hera is not considered as a separate design phase (or as a part of only one design phase), but should be addressed throughout all design steps. That is to say, different kinds of adaptation concerning the application's underlying data, its navigation structure, and presentation layer are foreseen. Furthermore, these adaptations should consider both the user (his preferences, characteristics, and navigation history) as well as his usage context (e.g. client device). However, prior to the work presented in this thesis, Hera's presentation model was not formalized, nor was adaptation addressed at presentation design.

Since parts of the work presented in this dissertation has been carried out within the scope of a collaboration with the Hera project (and especially the adaptive presentation model of Hera was designed, formalized in RDF(S), and implemented as an extension of the original Hera models in the context of this work), a detailed description of the Hera design phases and their corresponding component-based realization will be given in Chapter 5.

[12]Note that a definition for different types of hypermedia adaptation was provided in Section 2.2.

3.4 Discussion

This chapter provided an overview of related work on the field of engineering adaptive Web applications. First, it investigated component-based and document-centric solutions focusing on the presentation and implementation aspects of Web and multimedia applications. Second, it gave a summary of existing model-based design methods and methodologies aimed at the high-level specification of different design concerns involved in Web applications. In both cases, an important focus was put on the question whether (and how) the examined solutions support different kinds of adaptation, such as personalization, device and/or context dependency, etc.

The analysis of component-based and document-centric approaches has shown that there is a need for formats that abstract from the current coarse-grained implementation model of the WWW, thus facilitating to compose Web applications from fine-grained, declarative, reusable, and configurable implementation entities. In combination with an appropriate visual authoring tool, such a solution can be efficiently utilized in different application scenarios: from traditional hypermedia systems to more specialized multimedia (e.g. three-dimensional) or e-Learning applications. However, there is still a lack of approaches that explicitly focus on a clear separation of application concerns in different component types or levels. Moreoever, current declarative component models do not support the adaptation of reusable declarative implementation entities to different users, devices, and contexts in a component-based manner, nor do they provide an automatic presentation generation facility to various Web output formats. Furthermore, only a few of the mentioned solutions provide visual authoring tools that would support authors to proceed in an intuitive way based on a structured authoring process. Note that a detailed comparison of the approaches investigated in this thesis was already provided in Section 3.2.10.

On the other hand, the main strength of model-based design approaches is their support for the high-level design of Web applications in a structured and disciplined way. Based on the principle of separation of concerns, they facilitate to address a number of independent design issues and to express them in form of implementation independent high-level models. While there exist different formalisms and notations to express those models (e.g. UML, XML, RDF, etc.), an emerging trend is the application of Semantic Web languages for this purpose. The advantage of such approaches is a more explicit description of application semantics, resulting in better interoperability and (possibly) model verification support, as well as the possibility to integrate data (models) from different ontologies and sources. As discussed above, some of the existing methodologies provide support for selected kinds of adaptation, mostly at navigational design. Still, important adaptation issues, such as content-level adaptation (e.g. media adaptation) or (dynamic) adaptation at presentation design have not been sufficiently addressed, yet. In general, the explicit design support offered for the presentation aspects of a Web application is often neglected, as "most of the methodologies refer to templates (for example XSL templates) that describe the styling information of the systems" [Frasincar 2005]. Furthermore, even though some methodologies provide a (semi-)automatic code generation, fine-granular design artefacts get often lost during the implementation phase while being transformed to the current coarse-grained Web implementation model.

Finally, according to our best knowledge, there has been only one attempt to combine the benefits of Web design models with the advantages of component-based reuse at implementation level. As discussed in Section 3.3.2, that approach of Segor and Gaedke aimed at offering a number of heuristic implementation rules to map OOHDM design models to WCML components [Segor and Gaedke 2000]. However, neither was an automatic mapping

process achieved, nor were any kinds of hypermedia adaptation considered. Inspired by the component-based approaches presented in Section 3.2, the next chapters of this thesis will present a component-oriented document model as well as a corresponding visual authoring tool for adaptive Web applications. Furthermore, it will be also shown how an implementation based on this component-based format can be automatically generated from high-level design model specifications, thus adding automation to the overall process of design and implementation.

